パソコンプログラマのためのクリエイティブマガジン

月刊プログラマーズ・ページ

平成3年4月2日第三種郵便物認可　1991年8月1日発行(毎月1回18日)　第1巻第8号通巻8号

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

1991 8 AUGUST

定価980円



特集

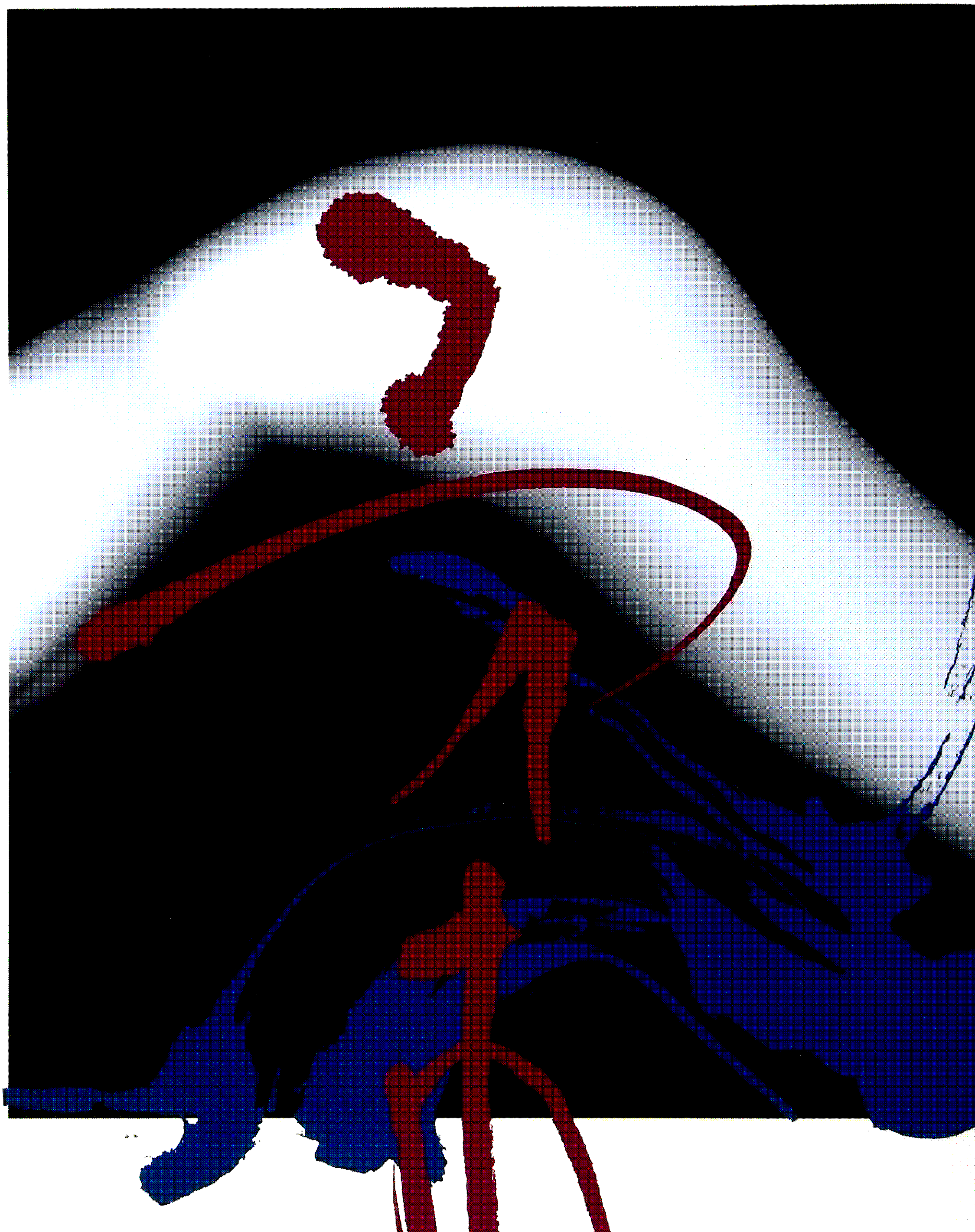
はじめてのオブジェクト指向プログラミング

オブジェクト指向な…分析・設計，プログラミングとは？

付録：5"2HDフロッピーディスク

本誌掲載…綴じ込み付録テキストファイル
(C++文法…ズ・ランダムスクリーン 解
読ツール …イバ選択プログラム
CINIT.SYS／通信ソフトWTERM

SE
Shoeisha Co., Ltd.



柔らかさを追求する
パーソナルコンピュータはAXです。

BORLAND
The Leader in Object-Oriented Programming

問：Windows™ アプリケーション開発に最も有効なコンパイラは、何ですか？

（解答はページ下をご覧下さい）

(Inactive C:¥WORK¥BORLAND.EXE)

	従来の開発環境	?
Cコンパイラ	¥ 98,000	¥ 98,000
C++コンパイラ	なし	含む
Windows AP作成ツール	¥ 98,000	含む
アセンブラ	¥ 40,000	含む
合　計	¥236,000	¥ 98,000

※上記の価格には、消費税は含まれていません。

パソコン上に、視覚的で快適な操作環境を提供するWindows 3.0。そこで利用するアプリケーション作成のためのコンパイラは、現在のものでベストといえるでしょうか？機能、スピード、価格…。これらの面で理想的ともいえるコンパイラが、登場しました。「BORLAND C++」です。BORLAND C++は、Windowsアプリケーションを開発、デバッグするC/C++コンパイラとツールを標準で装備しています。

■Windowsアプリケーション開発をサポート

BORLAND C++は、Windowsプログラムを完全にサポート。高速かつ高性能な処理を実現しています。MDI(Multiple Document Interface)やDLLs(Dynamic Link Libraries)といったようなWindowsの機能を使って、大規模なプロテクトモードアプリケーションの開発が可能になります。さらに、MicrosoftのリソースコンパイラとWINDOWS.H、Resource Work Shop(日本語版)が標準装備されていますので、別途の開発キット(SDK)は不要です。

■ANSI C／AT&T C++両用コンパイラ

BORLAND C++は、CとC++両方のコンパイラとして利用できます。さらに、再コンパイルの速度を大幅に向上させる「コンパイル済みヘッダ」機能をサポート。MS-DOSの640KBというメモリの制限を破るオーバーレイマネージャVROOMの利用や、Turbo Driveによってプロテクトモードで動作する統合開発環境での大規模なプログラムの作成も可能です。

■Programmer's Platform

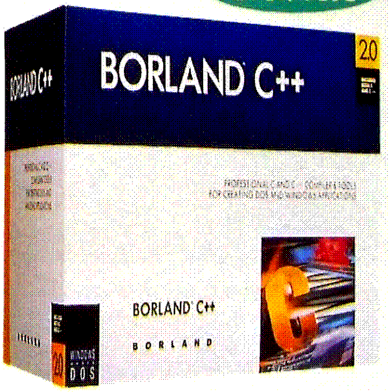
Programmer's Platformは、好きなツールを組み込む機能を持つ強力な統合開発環境で、MS-DOSやWindowsアプリケーション開発に必要なすべてのツールがすでに組み込まれています。さらにオーバーラップウインドウやマウス、そしてWindowsのプログラミングリファレンスを含んだオンラインヘルプをサポート。内蔵のマルチファイルエディタは、アンドゥ(実行の取消)／リドゥ(再実行)やマクロ機能を装えています。

■動作環境：PC-9801シリーズ(ハイレゾモード対応、LTは除く)／メモリ：640KB以上／MS-DOS：Ver 3.0以上／MS-Windows 3.0以上／マウス：バスマウス／要ハードディスク

Turbo C 2.0／Turbo C++1.0
登録ユーザの方には8月初旬に
アップデートのご案内をお送りします。

価格……………………98,000円（税別）

8月末発売



（答）BORLAND® C++

●製品に関するお問い合せは、総販売元㈱マイクロソフトウェアアソシエイツ TEL.03-3486-1411までお願いします。

株式会社ボーランドジャパン
〒107 東京都港区南青山7-8-1 小田急南青山ビル TEL.03-3486-1400㈹

＜資料請求番号001＞

i80386／486 32-bit専用DOSエクステンダー

DOS/4G



▲386ネイティブ・ソースレベル・デバッガ標準装

4ギガバイト・メモリ空間のアドバンテージ

32-bitパフォーマンスをDOS上で実現！

DOS/4Gは、Intel80386/486CPUが持つ32-bitパフォーマンスを、DOSプログラムからダイレクトに活用可能にする、80386/486CPU専用DOSエクステンダです。DOS/4Gにより、DOSプログラムで最大4ギガバイトまでのメモリが使用可能になります。また、32-bitレジスタを使用してセグメント切り替えの無いフラットなメモリ・アクセスが実現されるため、従来の16-bitプログラムに比較して約50%アクセス・スピードを向上できます。

DOS/4Gが組み込まれたアプリケーションは32-bit・ネイティブ・プログラムでありながら通常のDOS環境でそのまま動作します。しかも、WINDOWS等他のメモリ拡張手段に比べて、既存DOSアプリケーションからの改変作業は最小限で済ませることができます。32-bitCPUのメモリ・パフォーマンスをDOS/4Gでフルに活用することによって、従来のパソコン用アプリケーション・プログラムのパフォーマンス向上だけでなく、ワークステーション上の大規模プログラムのパソコン上への移行が可能になります。

■DPMI規格に完全準拠

● 80386プロテクトモードとDOSとのインターフェース標準規格DPMI(DOS Protected Mode Interface)に完全準拠しています。

● 従って、DOS/4Gで作成されたプログラムは、リアル・モードDOS環境下以外でもWINDOWS 3を始めとするDPMI規格準拠のホスト下で動作、または、共存が可能です。

■32-bitネイティブ・モード専用 高機能ソース・レベル・デバッガD/32を装備

● 32-bitアプリケーション EXE ファイルを、80386／486ネイティブ・プロテクトモード上でデバッグするための、ウインドウタイプ高機能デバッガです。

● DOS/4Gで作成された32-bitプログラムが、ソース・レベルでCodeViewと同様な感覚でデバッグできます。(右上画面写真参照)

■32-bitネイティブ・モード専用リンカーGLU/32を装備

● マイクロソフトLINKと互換性を保ったスクリプト・ファイルや、オプションが使用できます。

■仮想メモリ・マネージャVMM(別売)を用意

● DOS/4Gのオプション製品であるVMMは、パソコンに実装されている以上のメモリ空間を使用するアプリケーションの実行を可能にします。

● VMMは80386／486のページング機能を活用し、コードとデータを自動的にディスクへスワップさせます。

● VMMのAPIは、業界標準DPMI 1.0に含まれる仮想メモリAPIのスーパセットであり、上位互換性があります。

■業界標準のDOSエクステンダ

● 16-bitDOSエクステンダである「DOS／16M」は、Lotus、IBM、Ashton-Tate、DEC、AutoDesk、InformiXなど、300社をこえるソフトウェア開発会社の主要製品に用いられ、すでに数百万コピーのアプリケーションに組み込まれて出荷されています。

● DOS／4Gは、このDOS／16Mの上に構築されており、32-bitネイティブ・アプリケーション開発のためのアップグレード・パスとなります。

32ビット・16ビットDOSエクステンダ製品仕様

	新発売 DOS/4G	DOS/16M
対応機種	▶PC-9801 ▶PC/AT及び互換機 (但し、それぞれ80386以上搭載機)	▶PC-9801 ▶FMR60/70 ▶B16H×II/32GX ▶if800/RX,EX ▶PC/AT及び互換機(但し、それぞれ80286以上搭載機)
対応言語	▶WATCOM C 386 ▶High C 386 ▶Microsoft アセンブラ	▶Lattice C ▶Microsoft C ▶Microsoft Fortran ▶Microsoft アセンブラ
動作環境	▶MS-DOS V3.1以降 ▶MS-WINDOWS 3.0以降	

価格表	シングル・ライセンス	コーポレート・ライセンス
DOS/4G	¥880,000	¥2,640,000
DOS/4G用VMM	¥580,000	¥1,740,000
DOS/16M	¥780,000	¥2,340,000
DOS/16M用VMM	¥480,000	¥1,440,000

株式会社 LIFEBOAT
〒101 東京都千代田区神田錦町3-6
PHONE:03-3293-4711(代) FAX:03-3293-4710

※複数マシン(最大20台)で開発を行う際にお得なコーポレート用ライセンスも用意しております。

※製品についてのお問い合わせは……㈱LIFEBOAT 営業1課 DOSエクステンダー担当(Tel.03-3293-4714)までご連絡ください。

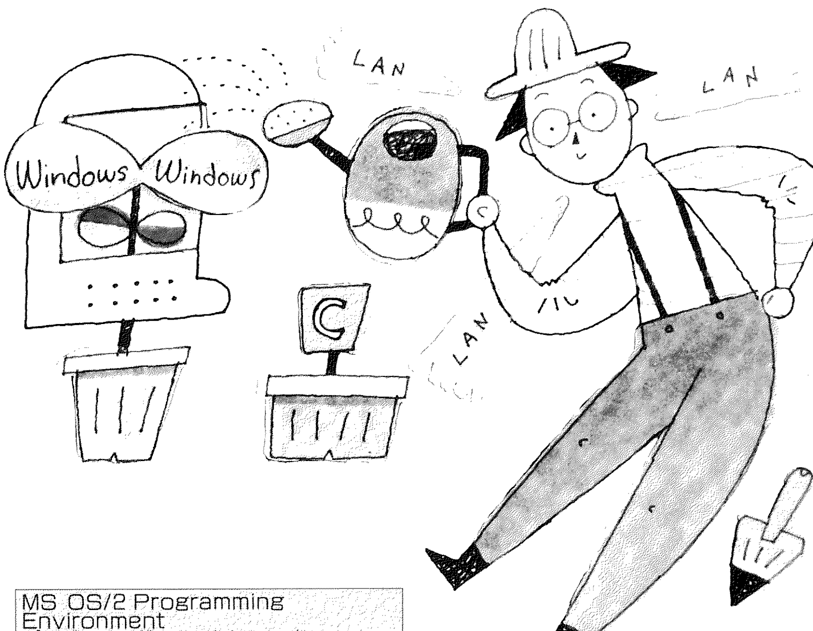
<資料請求番号002>

次代のアドバンテージ、最新の技術をあなたに！

Microsoft® University

独学より大学。

マルチメディアのプラットフォーム MS-Windows、高信頼性の MS OS/2。本格的なマルチタスクの実現、グラフィカルなユーザ・インタフェース（GUI）、LANマネージャによる高度なネットワーク処理など、技術者の創造力を刺激する環境を提供します。マイクロソフト・ユニバーシティは、その登竜門。ウインドウ環境、デバイスドライバの開発、ネットワーク処理など、アプリケーション・ソフト開発に欠かせない、ノウハウをレクチャします。入門から専門まで、充実したコースラインアップ。短時間で即開発に結びつく新しいコンピューティングスタイルを身につけて下さい。



Microsoft Windows Programming Environment
ウインドウズ・プログラミング・エンバイラメント

■期間：5日間 ■受講料：15万円（税別）

■MS Windows Ver3.0のアプリケーションを作成する方のためのコースです。Windows プログラミングのポイントを重点的に解説、実習中心のコースです。

Microsoft Windows Applications Programming
ウインドウズ・アプリケーションズ・プログラミング

■期間：5日間 ■受講料：18万円（税別）

■MS Windows Ver3.0対応のアプリケーションを開発するプログラマや設計者のためのコースです。ユーザインタフェースやアプリケーションなど設計ポイントがマスタできます。

Microsoft LAN Manager Programming Environment
LANマネージャ・プログラミング・エンバイラメント

■期間：4日間 ■受講料：15万円（税別）

■マイクロソフト LAN マネージャ（LM）用アプリケーションを開発する設計者のためのコースです。LMのアーキテクチャと特徴について解説します。

Microsoft LAN Manager Applications Programming
LANマネージャ・アプリケーションズ・プログラミング

■期間：5日間 ■受講料：18万円（税別）

■マイクロソフトLANマネージャ（LM）上で動作するアプリケーションの作成や移植するプログラマのためのコースです。

MS OS/2 Programming Environment
プログラミング・エンバイラメント

■期間：4日間 ■受講料：12万円（税別）

■MS OS/2のアプリケーションを作成するために必要となる基本的技術を修得するコースです。本コースは、OS/2に取り組む上での基本コースとなります。

MS OS/2 Presentation Manager Programming Environment
プレゼンテーション・マネージャ・プログラミング・エンバイラメント

■期間：5日間 ■受講料：15万円（税別）

■MS OS/2プレゼンテーション・マネージャのアプリケーションを作成するための入門コースです。作成に必要となる基本事項を説明いたします。

MS OS/2 Applications Programming
アプリケーションズ・プログラミング

■期間：5日間 ■受講料：15万円（税別）

■MS OS/2のAPIを使用し、マルチタスク、メモリ管理、プロセス間通信、ダイナミックリンクなどの機能を有効に活用する、実習を中心としたコースです。

C言語入門

■期間：2日間 ■受講料：6万円（税別）

■これからC言語を始めようとする方を対象に、C言語の基本的な文法からわかりやすく解説します。統合環境を持つQuick Cを使って理解を深めます。

MS OS/2 Presentation Manager Applications Programming
プレゼンテーション・マネージャ・アプリケーションズ・プログラミング

■期間：5日間 ■受講料：15万円（税別）

■MS OS/2 PMのアプリケーションを作成するためのステップアップコースです。経験者の方を対象に、より詳しく説明します。

MS OS/2 Device Drivers
デバイス・ドライバ

■期間：4日間 ■受講料：15万円（税別）

■MS OS/2のデバイス・ドライバの機能、構造を修得し、実際にデバイス・ドライバを作成するコースです。デバイスヘルパールーチンなども解説します。

C言語応用

■期間：3日間 ■受講料：9万円（税別）

■関数、ポインタ、構造体についてプログラムを作成しながら詳しく説明します。セミナは、最新のMicrosoft C Ver 6.0を使用して実施いたします。

●お問い合せは

富士ソフトウエア株式会社 MU教育センター

TEL.03-3452-7722（直通） FAX.03-3798-0599

日本電気株式会社殿の御協力によりPC-9800シリーズ対応コースも好評開催中!!

Microsoft®
マイクロソフト株式会社
〒160 東京都新宿区西新宿7-5-25 Kビルディング

FSI 富士ソフトウエア株式会社
MU教育センター
〒108 東京都港区三田3-4-3 三田第一長岡ビル

＜資料請求番号003＞

Brief

Full Screen Editor 日本語版

全米のプログラマが認めた実力！
アメリカでは「ブリーフ」が
プログラミング用エディターとして
最も多く利用されています。

C言語で組めるマクロ！

Version UP!

V.3.0 いよいよ登場。
プログラミングエディター新世紀。



よりパワフルで、よりエレガントなプログラミングのために……。

●C言語で組めるマクロ 新機能

使い慣れているC言語でマクロが組めますから、習得が簡単でカスタマイズも思いのままです。

●マクロのスクリーン・デバッガ 新機能

ソースコードでデバックできます。もちろんブレークポイントやウォッチ・ドッグもセットできます。

●テンプレート機能〈構文の自動生成〉新機能

C、Ada、Basic、COBOL、FORTRAN、MODULA2、Pascalで構文を自動生成します。

●300回ものアンドゥ

各ファイルごとに300までの操作を元にもどすことができますので、やり直しの面倒を気にせず、安心して編集できます。

●安全編集

Briefは、一定時間キーを操作しないこと最新の編集内容をファイルに自動保存します。だから編集中に電源が切れても安心です。

●その他の主な新機能

キー操作のファイルへのセーブとリプレイ、処理可能行数を拡張、リジューム機能、Ctrlキー・アサイン(オプション)、メニュー(オプション)etc。

アメリカでも日本でも高い評価

●プログラミング向きエディターを探している人は、もう探す必要はない。(Jerry Pournelle BYTE)
●ソフト開発には強力なツール、300回のアンドゥは大きな特徴。(日経バイト)
●既存のエディターに不満を覚えたら、これしかない。(Oh! FM)
●まるで夢のようなアンドゥだ——おもわず間違うのを期待してしまいそうだ。(The C Users Journal)

■主な機能

◆55ウィンドウ◆無限のファイル◆自動コンパイル◆自動エラー・ジャンプ◆自動インデント◆自動テンプレート◆バッファ書出 切換 削除 リスト表示◆ファイル名の指定 表示 変更◆バックアップ保存◆特殊文字入力◆2行を1行に連結◆ファイル単位の挿入◆増分検索◆各種マーク 始・終点の入換◆ブロックのカット コピー ペースト 印刷 保存 左右へのシフト◆対応する括弧へ移動◆C言語のコメント化 非コメント化◆DOSコマンドの実行◆キー操作記憶 実行◆入力ヒストリー◆英文テキスト整形◆タブのスペース挿入◆拡張子毎に異なった設定ができるタブ◆検索置換パターンの画面からの入力◆英字の大小文字変換 その他

定価 48,000円
(消費税は含まず)

■5機種対応パッケージ

(東芝、NEC、AX協議会、IBM、富士通)

●5 2DD、3.5 2DDデュアルメディア●MS-DOS3.1以降に対応●英語DOSでは使用できませんが、英語版BRIEFもセット販売いたしております。●お急ぎの方、上記以外のメディア希望の方、英語BRIEFのセット販売希望の方はユーザー・サポート係までご連絡下さい。

日本システム株式会社

●お問合せは　東京都立川市曙町1-18-2一清ビル立川事業所Briefユーザー・サポート係
TEL0425-23-4469　FAX0425-27-6127

＜資料請求番号004＞

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

特集
はじめてのオブジェクト指向プログラミング
オブジェクト指向な…分析・設計，プログラミングとは？

表紙イラスト：日暮修一
表紙AD：佐々木幹夫(SE)

パソコンプログラマのためのクリエイティブマガジン

月刊プログラマーズ・ページ

付録
5"2HDフロッピーディスク

綴じ込み付録
C++文法サマリ

1991 AUGUST No.8



Phar Lap Software, Inc.
386 | TOOLBOX

MS-DOSの640KByteという狭いメモリの制限を超えて、4GByteという広大なアドレス空間を利用者に提供するユーティリティ、それが386 | TOOLBOXです。
386 | TOOLBOXは、32bitプロテクトモードという従来のMS-DOSが十分に活用していなかった80386/486CPUの特長を、MS-DOSから100%活用可能にしました。
386 | TOOLBOXは、80386CPUをプロテクトモードに切り替えることによって、従来のMS-DOS環境のままでユーザプログラムがMS-DOSの640KByteのメモリ制限を破り、実装されているメモリを全て連続した記憶域として利用できるようにします。さらに80386CPU本来の実行速度や4GByteのアドレス空間を活用して、従来のMS-DOSプログラムより巨大なメモリを使用する実行速度の速いプログラムの作成・実行を実現しました。386 | TOOLBOXは、386 | DOS-Extender(DOSエクステンダ)のほか、386 | ASM(アンセブラ)、386 | LINK(リンカ)、386 | LIB(ライブラリアン)、MINIBUG(デバッガ)で構成されます。

●別売りオプション　仮想記憶マネージャ　386 | VMM
　シンボリックデバッガ　386 | DEBUG

MetaWare Inc.
HIGH C386

MetaWare社のHIGH C386は、80386CPUプロテクトモード用最適化コンパイラです。
HIGH C386は、80386CPUのプロテクトモード上で動作するハイレベルのCプログラムを構築するプロフェッショナルソフトウェア技術者のために設計されました。
正確なエラーメッセージと優れた特長を兼ね備えており、また、ソフトウェア開発者は多種多様なトグルスイッチとプラグマを活用して、各々の開発者のニーズに合わせたコンパイラの初期設定や仕様の変更が可能です。
HIGH C386は、386 | TOOLBOXに含まれる386 | DOS-Extenderとの組み合わせにより、80386CPUプロテクトモードの膨大なアドレス空間と高速な演算処理速度を活用できるようになります。

ベンチマークテスト
WHETSTONE(自社計測)
使用機器：PC-9801RL20MHz＋数値演算コプロセッサ(80387 or Turbo-3167)

emulator + noFPU	58
emulator + 80387	200
80387 inline code	1540
Turbo-3167 inline code	2380

(単位：K Whetstone)

価格

すべての価格に消費税は含まれていません。

386	TOOLBOX	¥148,000	NDP FORTRAN386	¥148,000
386	VMM	¥ 86,000	NDP C386	¥148,000
386	DEBUG	¥ 58,000	HIGH C386	¥148,000

MicroWay Inc.
NDP FORTRAN386

NDP FORTRAN386は、FORTRAN77仕様に完全準拠した80386CPUプロテクトモード専用FORTRANコンパイラです。
最大の特長である、プロテクトモードの4GByteのアドレス空間と、最適化による高速コードの生成により、VAX8600/SUN3といった汎用コンピュータ/ワークステーションクラスの規模と速度をもつ、アプリケーションの開発・実行を可能にしました。

特長
- ●プロテクトモードのアドレス空間4GByteを活用可能。1MByteを超えるプログラムや配列の実行・定義ができ、上限はパソコン本体に実装されたプロテクトモード用メモリの量にのみ制限されます。
- ●数値演算コプロセッサ80387をサポート。さらに、80387の約3倍(浮動小数点テスト時)の性能をもつ、Weitek1167/3167チップ、Turbo-3167ボードをサポート。NDP FORTRAN386で作成された計算ルーチンの速度をより向上させることができます。

動作環境

	MS-DOS	Coprocesser	Protectmode Memory
386	TOOLBOX●	要	対応	1MByte以上
386	VMM●	要	対応	1MByte以上
386	DEBUG●	要	対応	1MByte以上
NDP FORTRAN386	要	必須	2MByte以上
NDP C386	要	必須	2MByte以上
HIGH C386	要	対応	1MByte以上

	Harddisk	PC-9801/H98	PC/AT互換	FMR
386	TOOLBOX*	要	○	○	○
386	VMM*	要	○	○	○
386	DEBUG*	要	○	○	○
NDP FORTRAN386	要	○	○	
NDP C386	要	○	○	
HIGH C386	要	○	○	○

●印は、32bit機のみに対応します。
*印は、使用する機種別(NEC、IBM、FMR)に、購入していただく必要があります。

TSS
テックソフト&サービス株式会社
コンピュータ事業部　〒273 千葉県船橋市本町7-11-5 レランドセンタービル5F
Tel.(0474)23-1922　Fax.(0474)23-0202

特集

はじめての オブジェクト指向 プログラミング

オブジェクト指向な…分析・設計，プログラミングとは？

1990年代には，いままでの構造化プログラミングに代わって，オブジェクト指向プログラミングが新しいプログラミングパラダイムとして定着するといわれている．しかし，日夜プログラミングに汗を流す現場のプログラマにとって，オブジェクト指向というのはどうも得体の知れないしろものである．耳障りのいい話はいろいろ聞くものの，その実体がつかめないのだ．オブジェクト指向の必要性に疑問を感じるプログラマも少なくはないだろう．

本特集では，まずオブジェクト指向プログラミングとオブジェクト指向プログラミング言語に関する基礎的な概念を明確にするとともに，混乱のもととなる専門用語の整理・解説を行う．次に，STI(Software Technology International)でオブジェクトオリエンテッドテクノロジーのインストラクターを務めるアリス・サルペーター女史によるオブジェクト指向アプローチの実際についての解説を行い，C++の総本山ともいうべきUSL（UNIX System Laboratories Pacific. Ltd.）の阿部氏より，C++の有用性，特徴，将来，導入の実際について述べてもらう．また，Windows，Macintoshにおけるオブジェクト指向についてそれぞれ解説し，最後にクラスライブラリを含む安価なオブジェクト指向プログラミング言語であるTurbo Pascal 6.0の紹介を行う．



オブジェクト指向概念の基礎

オブジェクト指向パラダイムとプログラミング言語の実際

神林靖

いま話題のオブジェクト指向プログラミングだが，オブジェクト指向という考え方自体は，決して新しいものではない．その源流は，1960年代に開発されたSimulaまで遡ることができる．そして，このオブジェクト指向を純粋に昇華させたとでもいうべき言語が，1970年代にXerox-PARCで開発されたSmalltalkであった．以後，オブジェクト指向は既存のあらゆるプログラミング言語に影響を与え，数多くのオブジェクト指向プログラミング言語が生まれることになる．
コンピュータサイエンスの視点でオブジェクト指向プログラミングを見るならば，そこにはアクタ理論がその理論的背景として存在している．しかし，オブジェクト指向はもともとプログラミングの現場から生まれてきた考えであり，純粋数学の世界で生まれた理論をもとにしているわけではない．このため，オブジェクト指向のすべてをアクタ理論で説明しようとしても，やはり無理がある．また，オブジェクト指向といいながら，人によって概念が異なるなどという混乱も生じるわけである．そこで，オブジェクト指向の基礎概念を解説するとともに，オブジェクト指向に関する用語の整理を行ってこのような混乱の減少に努めた．

オブジェクト指向プログラミング

情報処理の世界で「オブジェクト指向」という言葉がひんぱんに使われるようになってきた．オジェクト指向プログラミングに対する関心も高い．1970年代に発生し，1980年代を通じて培われてきたプログラミング言語設計の方法論または計算モデルとしてのオブジェクト指向という考え方が，ここへきてシステム分析やシステム設計，あるいはデータベース設計へと応用が広がってきたことにもよるようだ．その反面，オブジェクト指向は80年代における構造化プログラミングのようなもので，誰もが援用していると主張するが，何がオブジェクト指向だか誰もわかっていないともいわれている．事実アカデミックな世界でも計算機科学者が10人いれば10とおりのオブジェクト指向があるともいわれるくらいに意見が分かれているのが現状である．そこで，筆者も主観的立場を前面に出して論を進めることにするが，その前にオブジェクト指向プログラミングとオブジェクト指向プログラミング言語の違いについて触れておきたい．

オブジェクト指向プログラミングとは，あるソフトウェアシステムをその構成要素となるオブジェクトに注目して，オブジェクト間の相互関係にもとづいて設計し実現するプログラミング活動である．そして，その実現を容易に行うための機能を備えた言語をオブジェクト指向プログラミング言語といい，それぞれ独立の存在である．したがって，たとえばCのような伝統的なプログラミング言語でオブジェクト指向のプログラミングを行うこともある程度可能であるし，Smalltalkのようなオブジェクト指向プログラミング言語を用いて非オブジェクト指向プログラミングをすることも可能である．

本稿では，オブジェクト指向プログラミングの基本概念をオブジェクト指向プログラミング言語を具体例として論を進めたい．

オブジェクト指向計算

オブジェクト指向プログラミング，あるいはより広義のオブジェクト指向計算とは何かについてまず考えてみたい．

オブジェクトとは「対象」と訳されることが多いが，要は「もの」のことである．人間が外部世界を理解する場合，それはあらゆる「もの」を通じてなされる．プログラミングとはプログラミング言語を用いて外的世界を写し取り，計算機内部に抽象世界を構築することであるといえるが，従来のプログラムでは計算機アーキテクチャの制約もあって，データと手続きはそれぞれ別の実体であった．コンピュータのハードウェアを制御するという意味で能動的な手続きを，ただのビット列であるという意味で受動的なデータ構造に適用するのはプログラマの責任であり，特定のデータ型上で手続きが正しく動作することもプログラマが確認しなければならない．

これに対して「物理的イメージを基礎とした抽象化をプログラミングの世界で実現する」ことを可能としたのが，オブジェクト指向プログラミングである．そこではオブジェクトはもはや受動的な存在ではなく，それ自身の中に状態と状態を操作する手続きをもつ実体である．したがって，プログラマはごく自然にプログラムの対象を表現することができる．たとえば，プログラムの断片やデータの集まりがすべて宙に浮いている状態を想像されたい．そしてプログラムコードとそれが用いるデータの間，そしていっしょに用いられたり，似たような機能を実行するプログラムコード同士の間に糸が張られていると想像してみよう．これら全部を動かして糸の長さがもっとも短くなるようにすると，各データ構造の周りにそれを使用するルーチンが集まってくる．これこそがプログラムコードの自然な分割であり，あとに述べるクラスを形成するものである．

われわれは外部世界が「もの」から成り立っていると知覚し，またわれわれの脳は情報をあるまとまった「かたまり」へと再構成する習慣がある．オブジェクト指向プログラミングによって，われわれが自然に身につけている思考習慣を直接用いる道が開けたのである．

プログラミングは科学である．背景には厳密な理論的背景が存在する．関数型プログラミングにはラムダ計算，コンビナトリ論理，カテゴリ論などが，そして論理型プログラミングには命題論理や述語論理といったものが存在する．オブジェクト指向プログラミングにはアクタ理論といったものが存在するが，前二者と異なる点は，それらが計算機科学での応用を当初からの目的

とはせず純粋数学の世界で培われたものであるのに対し，アクタ理論は純粋に計算科学者の手によるところにある．

しかも，オブジェクト指向パラダイムは，実際の活動の中から生まれてきたものなので，すべてがアクタ理論で説明できるわけでもない．この点が，オブジェクト指向プログラミングやオブジェクト指向計算についてさまざまな議論が存在し，いまだ定説を見ない一因であろう．以下では，まずオブジェクトとは何か，その中には何が含まれているのかを具体的に述べたうえで，中心となるアクタ理論について述べる．そのあと具体例に立ち帰り，代表的なオブジェクト指向言語であるSmall talkとその産みの親であるSimulaを，そしてオブジェクト指向が栄えているもう一方のLISP世界からFlavorとCLOSに登場してもらう．C系のオブジェクト指向プログラミング言語について深く触れないのは筆者の偏見ではなく（多少はあるかもしれないが），本誌の4月号でC言語の特集を組んでいて，そこの「C言語の明日」に詳しいからである．

オブジェクト

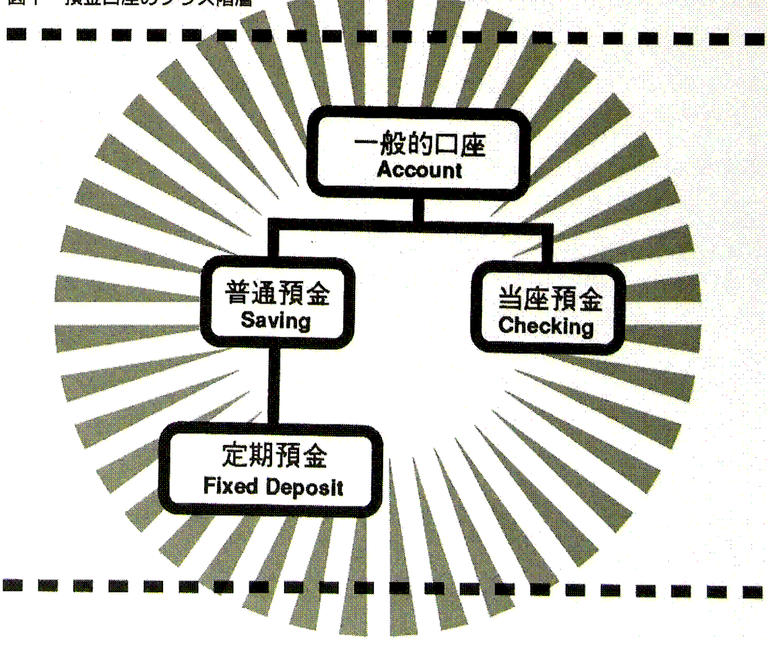
「オブジェクト」という用語は，1970年代を通じてXerox-PARCで開発された言語Smalltalkによってはじめて一般化したが，その概念は1960年代半ばに現れたSimulaの「クラス」構造まで遡れる．それはブロックのインスタンスを生成し保存するとともに，クラスの外側からポイントされているブロック中の変数や手続きを参照する手段を提供する方法でもあった．そのようなブロックの化身がクラスの「インスタンス」であり，オブジェクトと呼ばれた．一方，Smalltalkではすべてがオブジェクトであり，Simulaでオブジェクトと呼ばれるものは「インスタンス」と呼ばれる．実際Objectとは，すべてのクラスのもととなるクラスの名前として使用されている．

一般に，オブジェクトとは次のように定義できるであろう．すなわち，状態情報（データ）とそれに対する操作手段（手続き，メソッド）をもち，それらを単一の計算主体とするだけの制御能力をもった実体である．個々のオブジェクトは明確な境界をもち，特定の操作依頼手続き（メッセージ）を受け取ることによって自律的に行動する．行動は，オブジェクトがその時点でもつ状態と到着するメッセージによって決定される．オブジェクト指向プログラミングでは，このオブジェクトが分割される作業単位，システムの構成要素，並行動作するプロセス，再利用される部品などすべての単位となる．ある意味では，個々の手続きやデータにはある程度の再利用性を放棄させて，より高次のオブジェクトという単位でまとめることにより，システムの単純性統一性を高めようとしているともいえる．

メソッドとメッセージ

オブジェクトの状態を変化させ得るのは，そのオブジェクト内のメソッドのみであり，外部からはメソッドの起動を促すメッセージを送信できるだけである．メソッドとメッセージの区別は重要である．メソッドはオブジェクトの一部であり，大域的存在ではないため，同名のメソッドも送信されるオ

図1　預金口座のクラス階層



C++ & OOP 用語集
delete演算子　動的記憶割り当ての解放を提供する演算子で，標準のライブラリ関数であるfreeと似ている．

ブジェクトが異なれば起動される動作は自然と異なることになる．たとえば，Smalltalkでは「＋」というメソッドは，整数や実数のみならず座標にも定義されているため，同じ「＋」というメッセージでも送られるオブジェクトによってその動作対応は異なる．異なる対象が同名のメッセージに対して異なる反応を示すことはポリモルフィズムとして広く知られているが，オブジェクト指向プログラミングによる実現はもっともエレガントなものである．この機能を従来の命令型のプログラミング言語で実現する場合には，１つの手続き中にデータ型によって分岐する巨大な条件分岐命令を書かなければならない．

ポリモルフィズムによって，差分プログラミングとして知られるオブジェクト指向プログラミングの特徴が明らかになる．すなわち，すでに同名のメッセージに対して対応するほかのオブジェクトが存在していると，新しいオブジェクトをシステムに加えるのが容易になる．また既存のオブジェクトに新しいメソッドを追加する場合も，ほかのオブジェクトに対する副作用を考慮する必要がないので容易である．差分プログラミングについては，次に述べるクラスと継承の問題を導く．

クラス，インスタンス，継承

現代のプログラミング言語は，型定義と変数宣言を独立に扱っている．たとえばCでは，関数が呼ばれるたびに局所変数が実体化，つまり領域の割り当てと初期化が行われる．オブジェクト指向プログラミングでは型宣言に当たるものがクラスであるが，クラスは「オブジェクト」のところでも述べたように，局所変数としてのデータ構造とそれに対する操作を提供するメソッドをもつ．多くのオブジェクト指向システムではクラスは階層構造を採っているため，下位のクラスでは上位のクラスで定義されたデータ構造やメソッドを利用できる．これがクラスの継承といわれるものであり，オブジェクト指向システムをほかのプログラミングシステムから特徴づけている機能である．

銀行預金のシステムを考えてみよう．預金口座には大ざっぱにいって普通預金，当座預金，定期預金が考えられる．これらは，それぞれ性格が異なる．たとえば普通預金口座からは小切手を切れないし，定期預金は利率が高いが，それには一定期間の経過が必要である．それにもかかわらず基本的には，口座番号，預金者氏名，預金残高，利率といった属性を共通にもち，すべての口座に対して預金という動作を行うことができる．

従来のプログラミング手法では，各々に対応するデータ構造を定義し，それぞれにアクセスの手続きを定義しなければならない．いきおいデータ構造や手続きに重複が生じることになる．オブジェクト指向プログラミングでは，それがデータ構造であろうが手続きであろうが，クラス継承機能を用いて共通部分をくくりだすことができる．

預金口座のクラス階層を図１に，それをもとにしたSmalltalk-80による銀行預金のプログラムをリスト１に示す．共通の一般的口座としてAccount

リスト１　Smalltalkによる銀行預金のプログラム

```
クラス          Account
スーパークラス    Object
インスタンス変数  accountNumber balance rate
インスタンス・メソッド
deposit: amount
        balance ← balance + amount

withdraw: amount
        amount > balance ifTrue: [self error: "残高をご確認下さい"]
                         ifFalse: [balance ← balance - amount]

クラス          Saving
スーパークラス    Account
インスタンス変数  <none>
インスタンス・メソッド
check: amount
        self error: "小切手は当座預金でご使用下さい"

クラス          FixedDeposit
スーパークラス    Saving
インスタンス変数  period
インスタンス・メソッド
withdraw: amount
        (period mature) ifTrue: [↑super withdraw: amount]
                        ifFalse: [self error: "満期が来ておりません"]


クラス          Cheking
スーパークラス    Account
インスタンス変数  <none>
インスタンス・メソッド
check: amount
        self withdraw: amount.
        self issueCheck
```

C++ & OOP 用語集
new演算子　動的記憶割り当てを提供する演算子で，標準のライブラリ関数であるmallocと似ている．

というものが定義されている．実際にはこのような一般的口座というものが存在するわけではなく，便宜的に共通部分をもつクラスを定義しているに過ぎない．このようなクラスを抽象クラスという．たとえば哺乳類という動物はいないが，そのようなすべての哺乳類のもつ属性だけをもつ動物を便宜的に考えると動物の系統樹が理解しやすいのに似ている．普通預金口座Savingと当座預金口座Checkingは，一般的口座Accountのサブクラスである．また定期預金口座FixedDepositは，普通預金口座のサブクラスである．

アクタ理論

アクタ理論はオブジェクト指向計算の基礎となるものであり，オブジェクト指向の考え方をもっとも純粋な形で表している．そこにあるすべてのものは計算パワーをもって自律的に行動するアクタであり，すべての現象はアクタの生成とアクタからアクタへの通信によって表現される．共有領域，手続き呼び出しといったものがないため，並列処理を自然な形で記述できる．アクタには，本質的に並列性が内在しているということができる．アクタ理論でもっとも強調されるのは，メッセージ通信による制御フローとデータフローの統一，そして自律的計算である．この点は，オブジェクト指向プログラミングの特徴および今後の動向を考えるうえで重要なので詳しく述べてみたい．

リスト2　Lispによる階乗の定義

```
(defun factorial(n)
        (cond ((= n 1) 1)
        (t (* n (factorial (- n 1)))
        )))
```

リスト3　アクタによる階乗の定義

```
factorial=λm.
    match m<n  c>
     if n=1
         then (send c<1>)
else
     if n>1
         then (send factorial <(n-1)(λk.(send c<n * k>))>)
```

リスト4

```
(send factorial <3 answer>)
```

リスト5

```
(send factorial <2 (λk.(send answer <3 * k>))>)
```

リスト6

```
(send factorial <1 (λk'.(send (λk.(send answer <3 * k>))
                        <2 * k'>))>)
```

通常，手続き型言語では制御とデータは別々の流れをもつ．すなわち，計算は逐次的に行われるのに対し，計算結果であるデータはいったん変数に格納されたうえであとから必要になった時点で取り出される．このように，データは制御フローから独立している．
アクタ理論では，メッセージ送信だけが制御の流れを変える唯一の手段であるだけでなくデータをアクセスする唯一の手段でもある．データは，メッセージの一部としてあるアクタからほかのアクタへ送られる以外に授受の手段をもたないのだ．メッセージ送信は，データとともに制御の流れをほかへ移転するいう点で，手続き呼び出しや関数適用と似ているが，メッセージ送信はそれを受け取ったものが計算終了後に計算結果をメッセージの送信者に返還しないという点で大きく異なる．手続き呼び出しでは，呼び出す側と呼び出される側でどうしても主従の関係ができてしまう．ある部分作業を行ったサブルーチンは制御を主プログラムに返すわけである．

これに対しアクタでは，個々のアクタは自律的に行動するためアクタの数だけ制御フローがあり，メッセージ送信はそれらの制御フローに影響を与える，あるいは干渉するというふうに解釈することもできる．このメッセージ送信の単方向性によってアクタ間の対称性が維持され，並列計算の相互作用が自然に記述できるわけである．この

C++ & OOP 用語集
privateアクセス指定子　メンバはデフォルトでprivateであり，そのクラス内で宣言されたメンバ関数およびフレンド関数からのみアクセスできる．

リスト7

```
(send (λk'.(send (λk.(send answer <3 * k>))
                 <2 * k'>))
      <1>)
```

リスト8

```
(send (λk.(send answer <3 * k>)) <2 * 1>)
```

リスト9

```
(send answer <3 * 2>)
```

リスト10

```
(send answer <6>)
```

特徴は，アクタ理論が知識の自然で効果的な手続き的表現を研究するMITのPLANNERプロジェクトから発生したことを考えるとよく理解できる．

アクタ理論ではメッセージの到着をイベントというが，並列計算をイベントの半順序集合として表現することができる．アクタは必要な計算資源であるデータや手続きをもった自律的に行動する能動的な計算主体であるため，計算は完全に1つのアクタ内に閉じ込めることができる．

単方向のメッセージ送信によって計算が行われるようすを，よく知られた階乗の計算について見ることにしよう．nの階乗（n！）とは衆知のように，

$$n! = n \times (n-1) \times \ldots \times 2 \times 1$$

のように表せるが，比較のためにLISPによる表現をリスト2に示す．アクタによる定義をリスト3に示す．

このアクタ言語は現実のものではなく，インディアナ大学のフィルマンによってアクタ理論の解説のために考案された疑似言語で，この例も同著者の「協調型計算システム」より採録した．両者はよく似ているが，本質的な相違は再帰定義の記述にある．LISPでは関数適用で計算が進められるため，(factorial (-n 1))という式は評価されると(n-1)!の値を返すことが保証されているので，(* n (factorial ...))によってn＊(n-1)!が計算できる．それに対しアクタの場合，メッセージ送信は単方向であるため計算結果は返されない．したがって(n-1)の階乗を計算した結果をどこへ送って欲しいかも，メッセージの一部として(n-1)!を計算するアクタへ渡さなければならない．これが継続点で，リスト中answerで表されるアクタである．

例として，3の階乗を計算してみよう．メッセージは〈3 answer〉で，これをfactorialへ送信する（リスト4）．

これはリスト5のように展開される．

n=2の場合，さらにメッセージがfactorialへ送られる．わかりやすいようにラムダ束縛されている外側の変数にはプライム記号がつけてある（リスト6）．

factorialはn=1のとき，1を継続点へ送る（リスト7）．

〈1〉を継続点へ適用すると，外側のλk'によって束縛されたすべてのk'は1で置き換えられる（リスト8）．

〈2＊1〉の評価結果をさらに継続点へ送ると，その結果はリスト9のようになる．

つまりリスト10のように，結局アクタanswerにその要素が6であるメッセージが送られて階乗の計算は終了する．

この例でもわかるように，アクタの振舞いはラムダ計算によって記述できる．この点は注目されてよい．つまり，アクタについての議論をラムダ計算を用いて行うことができ，アクタによって記述されたプログラムを数学的に取り扱うことが可能だということである．これは並列計算のプログラムの正当性を考慮する際にきわめて重要である．

SimulaとSmalltalk

オブジェクト指向が追求され，もっとも成功した領域にプログラミング言語の設計がある．事実オブジェクト指向が世に出た最初はプログラミング言語Simulaとしてであり，アクタ理論より古い．Simulaでの経験がアクタ理論の中へ取り込まれていったと考え

るのが正しい．SimulaとSmalltalkがアクタ理論の表現と際だって異なる点は，それらが実用的なプログラミング言語であるというところにある．PLANNERプロジェクトからはアクタ理論の実現として多くの言語が開発されているが，いずれも研究室レベルにとどまっている．ひとつの例外はSchemeであるが，Schemeはアクタのように並列性を強く意識していない．

Simulaは，Algol60をモデルとしたシミュレーション言語である．文法はほぼAlgol60を踏襲しているが，前述のようにAlgol系の言語のブロックが計算終了と同時に消滅するのに対し，Simulaではインスタンスが計算終了後も存続できるようにクラス概念を導入した．このクラスには結合という機能が備わっていたために，クラスAに新たな処理を追加してクラスBを定義できるようにもなっている．さらに通常の主従関係のある手続き呼び出しと異って，2個の手続きが対称的に相手を呼び合うコルーチン呼び出し（疑似並行処理）をもつ．これらによりシミュレーションに必要な外界の事物を計算機内に強力に表現できるようになった．つまりSimulaは，クラス，インスタンス，クラス継承(Simulaではクラス結合という）といったオブジェクト指向の重要な概念を最初に導入した手続き型言語なのである．Smalltalkが提供している機能のほとんどは，多重継承を除いてSimulaでも実現されているといっても過言ではない．

一方，Smalltalkは，クラス，インスタンス，クラス継承の概念をSimulaからひき継ぎ，それらをオブジェクトという単一概念で統一して美しい世界を構築したのが最大の功績であるといえる．事実1970年代前半に開発がはじまった当初は，クラス自身はオブジェクトではなかった．

Smalltalkの特徴は，(1)すべてがオブジェクトであること，(2)計算はメッセージ交換によって行われること，の2点である．アクタ同様にデータ構造，制御構造はともにオブジェクトの中に閉じ込められていて，オブジェクト間の制御はメッセージ交換によってのみ行われる．またあらゆるオブジェクトは，なんらかのクラスのインスタンスである．すべてがオブジェクトであるので当然クラス自身もオブジェクトである．それではクラスのクラスは何かというと，そのインスタンスがクラスとなるようなメタクラスと呼ばれるクラスがすべてのクラスについて1つずつ存在する．こうすることによって，Simulaのプログラムが手続きの定義とクラス定義の混在であったものが，Smalltalkのプログラムはクラスの集合として表現されるようになった．このような概念的な美しさがSmalltalkの特徴であり，Simulaやほかのオブジェクト指向言語と分かつところである．

ここでメッセージ交換というのは，Smalltalkではアクタ理論におけるメッセージ送信と異なり，メッセージの送信は制御とデータを受信オブジェクトに渡したあと，受信オブジェクトの計算終了を待ってメッセージ送信式が計算結果に置き換えられることをいう．この点がアクタ理論で表現される制御とデータを渡すだけの単方向のメッセージ送信と大きく異なる．ある意味では，Simulaのコルーチンを踏襲したものということもできよう．

これは，アクタがメッセージ送信による新しい計算理論の確立を意図したのに対し，Smalltalkが個人用ワークステーションのプログラミング言語として開発され，オブジェクトとの対話的操作を重視して並列計算に重きを置かなかったという設計上の選択による．近年盛んになってきている並列オブジェクト指向言語ではこの点を重視し，よりアクタ理論に忠実なモデルが採用されている．

FlavorとCLOS

実用性ということでいえば，LISPにオブジェクト指向を取り入れた言語がもっとも優れている．ここでは米国の東海岸と西海岸でそれぞれ開発され，覇権を争っているFlavorとCLOSについて見ることにしよう．言語設計の中にそれぞれ地域的な文化がかいま見えておもしろい．

Flavorは，MITの人工知能研究グループによって開発され商業的にも成功したシステムである（Symbolicsという大企業がこのグループから誕生した）．Flavor（フレーバ）というのは彼らによる独特の用語で，Smalltalkのクラスに相当する．Smalltalkのクラスでは概念の抽象化による階層構造とそれによる属性の継承が中心であるのに対し，フレーバでは個々のフレーバの結合による新たなフレーバ(クラス)の定義が中心となる．Smalltalkのクラス関係を系統樹に例えるならば，Flavorのフレーバの定義はアイスクリームのフレーバ（たとえばチョコレートとかストロベリ）を組み合わせて（たとえばダブルとかトリプル）新しいフレーバを得る作業に例えられよう．

リスト11にFlavorによる待ち行列

C++ & OOP 用語集
publicアクセス指定子　あらゆる関数からメンバを使用可能にする．structのメンバのデフォルト．

の定義を示す．Flavorではフレーバという名のユーザー定義のデータ型に属するデータをオブジェクトと呼び，整数やリストなど通常のLISPのデータ型と区別する．この点が，すべてをオブジェクトとして統一しているSmalltalkと大きく異なる．概念の美しさよりも実用性を重視する東海岸のハッカー気質が読み取れる．

一方CLOSでは，Smalltalk同様にメタクラスを導入することによって概念上の統一を達成している．すなわちすべてのクラスのメタクラスとしてstandard-classが存在するが，LISPに既存のデータ型についてはstandard-type-classという別のメタクラスを導入することによって，通常のLISPのデータ型もオブジェクトとして取り扱えるようにしているのだ．メタクラスが異なるということは，実際にはオブジェクト指向の外で処理されるということなのだが，見かけのうえだけでも基本データ型とオブジェクトを統合しているあたりが，西海岸の計算機科学者の美的こだわりと見ることもできる．

リスト11中のdefflavorがフレーバを定義しているところで，フレーバのインスタンスであるオブジェクトに対してメソッドが定義できる．リスト11では待ち行列に対する挿入，削除，空のチェックがdefmethodによって行われている．この待ち行列の要素に優先度をもたせるように変更を加えたものがリスト12である．クラス（フレーバ）の継承機能を用いているため，新たなフレーバの宣言と優先度に関するメソッドを定義するだけでよい．ここではスーパークラスはqueue 1つだけであるが，複数の名前を書くことによって多重継承を容易に実現できる．LISPの柔軟な記法の賜といえるかもしれない．

リスト11　Flavorによる待ち行列の定義

```
(defflavor queue
  ((a-queue nil))
  ())

(defmethod (enqueue queue) (an-elem)
  (setq a-queue (nconc a-queue (list an-elem))))

(defmethod (dequeue queue) ()
  (pop a-queue))

(defmethod (is-empty queue) ()
  (null a-queue))
```

リスト12　Flavorの継承を用いた優先度つき待ち行列の定義

```
(defflavor priority-queue
  ()
  (queue))

(defmethod (enqueue priority-queue) (an-elem)
  (setq a-queue (sort (cons an-elem a-queue) #'<)))
```

ここで，なぜ多重継承が必要な場合があるかを考えてみよう．図形を扱うプログラムを作成するとして，まず四角形一般の属性をもつクラスRectangleを定義する．そのサブクラスとして4隅が直角である長方形を定義するのは自然であろう．また，4辺の長さが等しい菱形も四角形のサブクラスである．それぞれのサブクラス内で定義された属性は，互いに独立である．さて，ここで長方形と菱形の両方の属性をもつような四角形のサブクラスを定義する必要がでてくる．すなわち，正方形の定義である．単一継承のシステムで正方形のクラスを定義するには，長方形のクラスのサブクラスとして定義して4辺が等しいという属性をつけ加えるか，菱形のクラスのサブクラスとして4隅が直角という属性をつけ加える．いずれにしても4辺が等しいという属性と4隅が直角という，それぞれ独立しているはずの属性間に不自然な上下関係ができてしまう．それだけでなく，インスタンス変数とそれを操作するメソッドの重複も起きてしまうであろう．この問題は，正方形のクラスのスーパークラスとして長方形と菱形の両方を許すようにすれば容易に解決する．事実そこにはつけ加えるべき属性は何もない．

このようにプログラムのモジュール化を徹底する手段として継承を考えるならば，多重継承が不可欠であることがわかる．Flavorでは，クラスといわずアイスクリームの組み合わせをいうように（実際にFlavorの開発者が多重継承を考えるときアイスクリームをイメージしたかどうかはわからないが）フレーバという言葉を選んだのは，スーパークラスに対して上位の抽象概念というよりは，このように組み合わせられる部品という見方を前面に出したかったからであろう．

多重継承によって先の問題が解決されたからといって，よいことばかりではない．単一継承ではクラス階層は木構造となるが，多重継承ではクラスの構造はグラフとなる．上位のクラスがさらに複数のスーパークラスをもった

C++ & OOP用語集
アクセス制御子　public，private，protectedの3つがあり，クラスのメンバのアクセス属性を制御する．

りするとクラス階層が崩れてしまう場合が出てくる. 実際Flavorでは, スーパークラスのメソッドが複数回起動されて暴走するといったこともあったらしい. またメソッドの結合にも順序付けのための型など, 実にさまざまなバリエーションがあって, 強力なのだが複雑でわかりにくい. この点CLOSでは, クラス順序リストによってクラスの継承関係を一元的に管理する方法を取っている. こういうところにも西海岸のやさしさを感じる.

まとめ

アクタ理論を実現した言語の多くがLISP系の記法を採用しているように, SmalltalkとFalvorやCLOSの間には見かけほど大きな差異はない. もっとも大きな差異は効率というものへの取り組み方であろう. Smalltalkの処理系も現在ではずいぶん早くなったが, それでも整数や実数といったものも含めてすべてをオブジェクトとし, すべての計算をメッセージの交換で処理しているためどうしてもオーバーヘッドが生じてしまう. Flavorではそこのところをわり切って, 基本データ型はオブジェクトとせず関数の適用ですましている. CLOSでも統合は見かけのうえだけで, 内部の処理は明確に分けている. C++では, Cの関数の枠組みの中でオブジェクト指向を処理しようとしているし, Objective-Cはオブジェクト指向の枠組みをCで表現しようとしている. これら既存の言語をなんらかの形で採用しているプログラミング言語に共通していえることは, 実行効率はさておきオブジェクト指向の思想から見るとどこか不自然だということである. Objective-Cはかなり成功した例であると思うが, それでもSmalltalkの美しさにはかなわない.

今後オブジェクト指向プログラミングが実用ソフトウェアへ採用されるにつれて, C++のような現実指向のプログラミング言語が主流になっていくのかもしれないが, それとともにオブジェクト指向の考え方に混乱が生じなければよいと思う. これからオブジェクト指向に取り組もうという読者には, 最初にSmalltalkから入られることを強く薦めたい. 言語は, 道具であると同時に思考を導くものでもあるからである.

冒頭で述べたようにオブジェクト指向プログラミングとオブジェクト指向プログラミング言語は独立のものなので, たとえばSmalltalkで非オブジェクト指向プログラミングもできるが, 実際にやってみると不自然なことが初心者にもわかる. 商業的に成功するには, Smalltalkによってオブジェクト指向の考え方とセンスを身につけたうえで, ほかの実用的な言語を使用して開発を進めるのがよいであろう.

ところで, オブジェクト指向を語るうえで創造的メディアの側面もわすれてはならない. そもそもSmalltalkの開発の経緯は, アラン・ケイが提唱した創造的メディアとしてのダイナブック（某大企業の製品ではない）のためのプログラミング言語としてはじまっているのだ. Smalltalkも, そしてそれと切っても切れない関係にあるマウスやビットマップディスプレイも創造的メディアの一部であり, 創造的思考を支援するという観点からオブジェクトとメッセージ送信が, アクタ理論からSimulaへと取り込まれる形でSmalltalkが完成したと見ることもできるのだ.

今後の動向としては, Objective-Cに代表されるような実用ソフトウェア開発のための知的道具としてのオブジェクト指向, Smalltalkに代表される情報や知識を蓄え, 操作するための創造的メディアを支援するためのオブジェクト指向と, そして並列計算を数学的厳密さで記述するためのモデルとしてのオブジェクト指向の3つが中心的柱になると考えられる. とりわけアクタ理論とラムダ計算が実現上同一視できるように, その親和性は重要であろう. LISP系の言語にみられる関数型との融合, ICOTのシステム記述言語にみられる論理型との融合にみられるように, 今後発展していく大規模並列処理の中で, アクタ理論そしてオブジェクト指向プログラミングの果たす役割は大きい.

参考文献

(1)山本喜一,「オブジェクト指向とC言語」, Computer Today, No.40, 1990.11

(2)神林靖, 石川雅基,「オブジェクト指向プログラミング」, コンピュータアクセス, 1989.10

(3)石川雅基, 神林靖,「C言語の明日－オブジェクト指向Cプログラミング」, プログラマーズ・ページ, 1991.4

(4)梅村恭司, 大里延康,「Lisp上のオブジェクト指向プログラミング」, 情報処理 Vol.129, 1988

(5) D.Thomas, " What's in an Object?", BYTE, March, 1989

(6) R.E.フィルマン, D.P.フリードマン, 雨宮真人, 尾内理紀夫, 高橋直久訳「協調型計算システム」, マグロウヒルブック, 1986

C++ & OOP 用語集
インライン関数　インライン関数は, クラスの中で宣言し他の場所で定義することができる. またクラス中でインライン関数の宣言と定義の両方を行うことができる.

特集
はじめての
オブジェクト指向
プログラミング

神林靖

オブジェクト指向プログラミングの用語解説

（五十音順）

インスタンス
(instance)

Smalltalkでは，クラスとして定義されたデータ構造と手続きからなる構造体の特定の事例で，Simulaのオブジェクトに相当する．同じクラスから生成されたインスタンスは共通のデータ構造と手続きをもつが，個別の状態（インスタンス変数）をもつことによる独自性は保つため，同じメッセージに対して同じ振舞いをするとは限らない．C++ではクラスというデータ型の変数を，Object Pascalではオブジェクト型の変数をインスタンスと呼ぶ．C++では共通のクラスに属するものをメンバと呼ぶため，インスタンス変数はデータメンバと呼ばれる．

インヘリタンス
(inheritance)

継承と訳される．Simulaではクラス結合（concatenation）と呼ばれた．上位のクラスの属性を下位のクラスが自動的にもつこと．具体的には上位のクラスで定義されたデータと手続きを下位のクラスが使用することができるが，通常データの型が継承されるだけで，データの実体そのものは異なる．アプリケーションプログラマは，上位のクラスで定義されたデータや手続きに類似のデータや手続きを定義する際，その差異のみを記述すればよくなるので，ソフトウェア資産が蓄積されるにつれて生産性が格段に向上する．これを差分プログラミングという．一方情報隠ぺいという観点からみると，サブクラスから上位のクラスの情報をアクセスできるわけで副作用を発生させる可能性もある．上位のクラスのメソッドを上書きしたい場合は，同名のメソッド名からなるメソッドを自分のクラスに定義すればよい．Smalltalkで特別に上位のクラスにある同名のメソッドを参照したい場合は，superと指定してメッセージを送ればよい．ちなみに自分自身にメッセージを送る場合はself（C++だけはthis）である．自分自身にメッセージを送ることにより，再帰定義が可能になる．ポリモルフィズム，多重継承も参照．

オブジェクト
(object)

状態とそれを操作する手続きをもった自律的な実体．個々のオブジェクトは明確に定義された境界をもち，外界からはその定義にしたがったメッセージを送ることによってのみ状態を変化させることができる．Simulaでは，クラスとして定義されたブロックのインスタンスがオブジェクトと呼ばれた．一方Smalltalkでは，すべてのものがオブジェクトとして定義される．Smalltalkでは，整然としたクラス階層の頂点に立つのがクラスObjectであり，計算機内で表現されるすべてのものに共通な属性がここで定義される．クラスもまたオブジェクトであり，SimulaのオブジェクトはSmalltalkではインスタンスと呼ばれる．ひるがえって，FlavorではFlavorと呼ばれるユーザー定義のデータ型のインスタンスをオブジェクトと呼んでLISPに既存のデータ型と区別している．それに対して同じLISP系の言語であるCLOSは，Smalltalk同様にメタクラスという概念を導入することによって通常のデータ型もオブジェクトとしている．C系の言語では，オブジェクトとインスタンスは同意語で，C++ではクラスというデータ型の変数を，Objective-Cではオブジェクト型の変数をオブジェクトと呼ぶ．

カプセル化
(encapsulation)

データと手続きを一体化させること．オブジェクト指向はカプセル化を追求する最良の方法の1つであるが，オブジェクト指向に限った概念ではない．構造化プログラミングに源を発する抽象データ型はカプセル化そのものである．Adaのパッケージは，抽象データ型

オブジェクト指向プログラミングの
用語解説

によるカプセル化を徹底したものといえる．

ガベージコレクション (gabage collection)

システムが提供する機能の１つ．ユーザープログラムがもはや使用しなくなったメモリ領域を自動的に検知して再利用できるようにする．オブジェクト指向プログラミング言語ではプログラムの実行時に動的なオブジェクトの生成廃棄をひんぱんに行うので，こういった機能があると便利である．ユーザープログラムからのメモリ要求に応えられなくなったときにまとめて回収する方法と，空き時間を利用してつねに少しずつ回収する方法とがある．PascalやC系のオブジェクト指向言語の処理系にはこの機能がないため，ユーザーが自らメモリ管理を行わなければならない．LISPの処理系には昔からあった．

クラス (class)

データの構造と振舞いの抽象的な記述．通常の型宣言と異なり，データ構造に加えてそのデータ構造を操作する手続きを合わせもつ．Simula, C＋＋, Objective-Cではオブジェクトを生成することのできるデータ構造．Smalltalkでは，データ構造はそのクラスのインスタンスすべての間で共有されるデータ構造であるクラス変数と，各インスタンスごとに固有なインスタンス変数からなる．クラス変数は大域データ，インスタンス変数は局所データと見ることもできる．Smalltalkにはこのほかに特定のクラスの間で共有されるプール変数というものがある．Flavorではクラスのことをフレーバ，Object Pascalではオブジェクト型とそれぞれ呼ぶ．C＋＋ではクラスのインスタンス間で共有される属性のことを一般にメンバ (member) と呼び，クラスは構造体とそれに関連付けられたメンバ関数から構成される．構造体はさらに公開部と非公開部にわかれ，非公開部はそのクラスに関連付けられたメンバ関数でのみアクセスできる．

クラスオブジェクト (class object)

Smalltalkには，Classという名前の特別なクラスがあり，そのインスタンスであるクラスオブジェクトが新しいクラスを生成するのに使用される．通常クラスの生成はブラウザ内にクラス定義を（既存のクラスのサブクラスとして）記述することによって自動的に行われるので，この特別なクラスをユーザーが意識することはないが，Smalltalkで多重継承を行おうとする場合は，クラスClassに対して複数のスーパークラスを引数としてもつメッセージを陽に送らなければならない．Objective-Cではファクトリオブジェクトと呼ばれ，クラスを定義することによって生成される．

クラス階層 (class hierarchy)

クラスの上下関係を体系化したもの．クラス階層にしたがってデータやメソッドは継承される．SimulaやSmalltalkの基本形での単一継承では木構造となるが，FlavorやCLOSなどの多重継承の言語ではグラフとなる (Smalltalkでも多重継承は可能だが，基本は単一継承である)．あるクラスからみて上位にあるクラスをスーパークラス (super class) 下位のクラスをサブクラス (subclass) という．スーパークラスのことを，C＋＋ではbase class，Object Pascalではancestor class，サブクラスのことは，それぞれderived class，child classという．

コンストラクタ (constructor)

C＋＋において定義されたクラスと同名のメンバ関数．クラスのインスタンスを初期化するのに使用される．不要になったインスタンスを消去するにはデストラクタ (destructor) を使用する．SmalltalkやLISP系の言語ではガベージコレクションが完備しているのでユーザーがメモリ管理に気を使う必要はない．

ジェネリック関数 (generic function)

CLOSで定義されるメソッド．Flavorも含めて通常のオブジェクト指向言語

C＋＋ & OOP用語集
オーバーロード関数　オーバーロードとは多重定義のことで，同一スコープ内で同じ名前で定義されている関数を指す．

オブジェクト指向プログラミングの用語解説

ではメソッドはクラスに属するものとして定義されるので，関数呼び出しの際の第1引数（Smalltalkでいうレシーバ）のクラス内のメソッドが起動される．これに対しCLOSのジェネリック関数は引数のべき集合に属すると考えられるため，呼び出しが起こったときのすべての引数のクラスによって動的に選択される．たとえば「3＋4」という演算は，通常のオブジェクト指向の考え方でいえば整数3に対して「＋4」というメッセージが送られ，その結果整数3が自分自身を7に変化させると考えられる．ジェネリック関数では，整数3と整数4の組に対して「＋」というメッセージが送られると考える．

情報隠ぺい (information hiding)

ソフトウェアの構成要素は，それを使うものがインターフェース情報以上の実現の詳細を知る必要がないようになっているのがよい．このような原則を情報隠ぺいといい，構成要素の連結の度合を減らすことによりソフトウェアの保守性や信頼性を高めることができる．オブジェクトやクラスは情報隠ぺいの優れた手段であるが，考え方そのものは構造化プログラミングの昔からあった．

多重継承 (multiple inheritance)

下位のクラスが上位のクラスの属性（データ構造とメソッド）を自動的にもつことを継承というが，すぐ上のクラスが必ず1つであるものを単一継承（single inheritance）といい，複数のスーパークラスが許されている場合を多重継承という．Simula，Objective-Cは単一継承，C＋＋，Flavor，CLOSは多重継承を認めている．Smalltalkは以前は単一継承であったものが，Smalltalk-80で多重継承も可能になった．しかし通常のクラスの作成はブラウザ内にその定義を記述すればよいのに対し，クラス・オブジェクトに特別のメッセージを送らなければならないなど，不自然さは拭えない．Smalltalkも基本的には単一継承と考えたほうがよいかもしれない．インヘリタンス参照．

抽象データ型 (abstract data type)

プログラムを管理可能な単位に分割してプログラムの複雑性を抑える手法は，構造化プログラミング以降広く流布しているが，その際個々のプログラム単位がどのように実現されているかを知らなくても，その抽象的な仕様や使い方がわかるというのはきわめて重要である．それによってプログラムの一部にデータ構造の変更やアルゴリズムの変更が起きても全体を手直しする必要がなくなる．このような抽象化の単位として手続きを採用したものが手続き抽象と呼ばれ，データ構造をも合わせもたせたものをデータ抽象と呼ぶ．抽象データ型はこれらをもう一歩進め，そのようにしてできたプログラム単位をあたかもデータ型のように使用する方法である．そこではデータ構造とそれを操作する手続きが一体化され，明確に定義されたインターフェース以外では外側からアクセスできないようになっている．C＋＋のオブジェクトは抽象データ型にかぎりなく近い．オブジェクトから（概念的な）計算パワーを抜き取ったものと考えることもできる．

動的束縛 (dynamic binding)

遅延束縛（late binding）ともいう．メッセージに対する手続きの起動が実行時に決定されることをいう．これに対して手続きがコンパイル時に決定されるものを静的束縛という．LISP系はもとよりSmalltalkやObjective-Cも動的束縛である．これは，同じ名前をもつメソッドが複数のクラスに定義されていることを考えれば自然である．C＋＋ではCにならって関数は基本的に静的に束縛されるが，メンバ関数をバーチャル（virtual）宣言することによってメッセージの動的束縛も実現できる．

部品化再利用 (reusability)

差分プログラミング（インヘリタンスの項参照）をさらに進めたもの．差分プログラミングでは，すでにある（完結した実行可能な）クラス定義の全部あるいは一部を継承して利用するが，

オブジェクト指向プログラミングの

用語解説

部品化再利用では，あらかじめ部品となるソフトウェア構成要素を用意しておきプログラム作成時にそれらを組み合わせることによりソフトウェアを完成させる．Objective-Cではこのようなソフトウェア構成要素をSoftware-ICと呼び，バイナリで提供するライブラリとしている．

ブラウザ (brawser)

Smalltalkのプログラミング環境の一部．Smalltalkのクラスライブラリはブラウザを通じてアクセスされる．これによりクラス階層の中を縦横無尽に探索することができる．ユーザーはブラウザを使って適当なスーパークラスを見つけて自分自身のクラスを定義する．クラス定義（通常の意味でのプログラミング）もブラウザ内で行われる．

ポリモルフィズム (polymorphism)

1つの変数が場合によって型の異なる値を取り得ること，あるいは1つの関数が場合によって型の異なる引数を取り得ることをいう．同じ関数名がいくつかの異なる関数で使用されるときにもポリモルフィズムと呼ばれる．これをとくに多重定義（over loading）という．通常メソッドはクラスに定義されているので，同じ名前が使用されていても衝突が起きることはない．たとえば「＋」というメッセージはそれを受け取るオブジェクトが整数か複素数かによって起動するメソッドが異なる．

メソッド (method)

クラス内のデータを操作するための手続き．Smalltalkではクラス内に定義されるが，LISP系の言語では別途に定義され名前で関連付けられる．通常の手続きや関数と区別して，とくにメッセージ送信の形で呼び出されるものに対して使われることが多い．C＋＋ではメソッドは通常の関数として実現されているが，Objective-Cでは通常の関数とは概念的に異なるものとして定義され，オブジェクトへのアクセスはすべてメソッドを介するようにしている．C＋＋でメソッドにあたるものはメンバ関数と呼ばれる．メンバ関数はクラスの中に定義され，そのクラスの非公開部をアクセスする．C＋＋にはメンバ関数のほかにフレンド関数と呼ばれるものもあり，これは特別にほかのクラスの非公開部のアクセスを許された関数である．このようにあるクラスからクラスのインタフェースを通すことなくアクセスできるクラスをフレンド（friend）という．

メタクラス (metaclass)

Smalltalkでいうところのクラスオブジェクトのクラス．すべてのクラスについて1つずつメタクラスが存在する．クラスオブジェクトは，このメタクラスの唯一のインスタンスである．メタクラスによってクラスの振舞いが決定される．ほかの多くの言語ではクラスをオブジェクトとしていないのでメタクラスは必要ない．

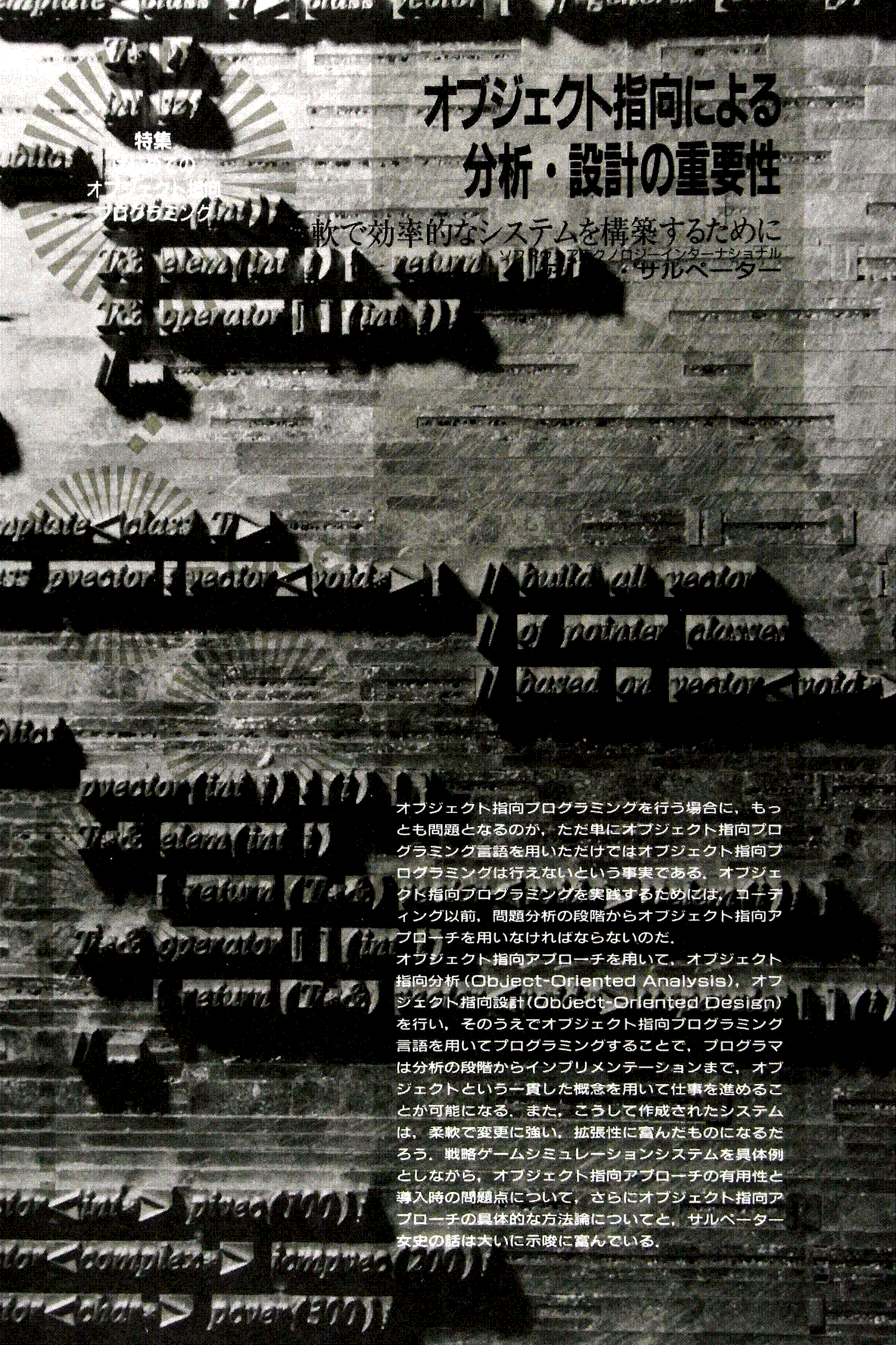
メッセージ (message)

オブジェクト指向の計算を起動させるもの．Smalltalkでは，あらゆる計算処理がオブジェクトからオブジェクトへメッセージを送ることによって行われる．メッセージが送信されると受け取ったオブジェクト内のメソッドが起動される．LISP系の言語やC＋＋では，メソッドの起動は通常の関数適用の形で実現されているのでメッセージ送信というメタファは前面に出ない．あえていえばメンバ関数の適用がメッセージ送信か．

ヨーヨー問題 (yo-yo problem)

あるメソッドがどのように実現されているかを見るために，クラス階層の中をくまなく捜しまわらなければならないこと．現実のプログラム開発では，オブジェクトによる情報隠ぺいですべてが片づけられるわけではなく，実現されているプログラムコードを解読しなければならないことも多い．そのような場合，多重定義やインヘリタンスは，プログラムの可読性を損なう場合がある．

C＋＋ & OOP用語集
仮想関数　仮想関数は派生クラスが基本クラスの関数と異なるバージョンを可能にする．仮想関数は基本クラス内で宣言し，引数が同じでも派生クラス内で再定義できる．



特集
[illegible]の
オブジェクト指向
プログラミング

オブジェクト指向による分析・設計の重要性

柔軟で効率的なシステムを構築するために

ソフ[illegible]テクノロジーインターナショナル
[illegible]・サルペーター

オブジェクト指向プログラミングを行う場合に，もっとも問題となるのが，ただ単にオブジェクト指向プログラミング言語を用いただけではオブジェクト指向プログラミングは行えないという事実である．オブジェクト指向プログラミングを実践するためには，コーディング以前，問題分析の段階からオブジェクト指向アプローチを用いなければならないのだ．
オブジェクト指向アプローチを用いて，オブジェクト指向分析（Object-Oriented Analysis），オブジェクト指向設計（Object-Oriented Design）を行い，そのうえでオブジェクト指向プログラミング言語を用いてプログラミングすることで，プログラマは分析の段階からインプリメンテーションまで，オブジェクトという一貫した概念を用いて仕事を進めることが可能になる．また，こうして作成されたシステムは，柔軟で変更に強い，拡張性に富んだものになるだろう．戦略ゲームシミュレーションシステムを具体例としながら，オブジェクト指向アプローチの有用性と導入時の問題点について，さらにオブジェクト指向アプローチの具体的な方法論についてと，サルペーター女史の話は大いに示唆に富んでいる．

オブジェクト指向テクノロジーの応用例
戦略シミュレーションシステム

まず最初に，私がイスラエル軍中央コンピュータセンターに勤務していたときに参加したひとつのプロジェクトについてお話したいと思います．このプロジェクトは，実際の戦争状態のなかで何かを判断しなければならないときの考え方を訓練するという，戦略ゲームシミュレーションシステムの作成でした．このシステムは，とても大きく複雑なもので，現実の戦争をよく反映しなければならなかったために，精度の高いアルゴリズムや高度なグラフィックス，マルチユーザー環境などが要求されました．

このプロジェクトでもっとも困難だったのは，このシステムに対する要求がいったいどんなものなのか，どうもはっきりわからなかったという点です．シミュレーションシステムに対する具体的な要求というものが，明確になっていなかったのです．

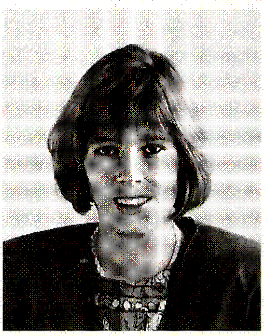
よいシステムを構築するためには，実際にそのシステムを使用するユーザーの協力が不可欠ですが，このときのユーザー，すなわちイスラエル軍の司令官たちは，コンピュータについては何も知らない人たちで，私たちシステムエンジニアがこのシミュレーションシステムに何を提供できるのか，明確に把握していないという状況でした．また，逆にシステムエンジニアは，コンピュータサイエンスについてはよくわかっているけれども，現実の戦争や軍の生活というようなことについてはまったくわかっていませんでした．

プロトタイプの作成

このため，私たちは一度で最終的なシステムを完成させるのはとても無理だと考え，まずこちらでプロトタイプのシステムを作って，それをユーザーに見てもらい，そのうえで最終的なシステムに必要なことがらを提案してもらおうと考えました．

そこで，まずプロトタイプのデザインをすることになったわけですが，ユーザーの意見を十分引き出すためにも，単なるモックアップのようなものではなく，小さくても実際に使えるレベルのものを作ることにしました．なにしろ，司令官たちはたいへん忙しくて，つまらないものに時間を割いてはくれません．彼らが，「これはいいものだ」と納得してくれるレベルのものを作らなければならなかったわけです．

実際にC言語とリレーショナルデータベースのオラクルを用いて，こういうかたちのプロトタイプを作るのに，15人年というたいへんな工数がかかりました．



Alice L. Salpeter M. Sc.

略歴

イスラエルのテルアビブ大学において，コンピュータサイエンスと経済学の学士過程を終了後，コンピュータサイエンスの修士号を取得．修士論文のテーマは，C++をベースにしたコンカレントなオブジェクト指向プログラミング言語の開発．実際のインプリメンテーションは，複数のプロセッサが動作するパラレル環境のもとで行われたという．

その後，イスラエル軍中央コンピュータセンター（Mamram）に情報システム担当将校として4年間勤務し，システムエンジニアとしてオブジェクト指向テクノロジーをもとにした，戦略ゲームシミュレーションシステムなどの開発を行うとともに，上級インストラクターとしてAI，グラフィックス，オブジェクト指向テクノロジー，C++などに関するプログラミング教育に従事した．

現在は，日本においてSTI（Software Technology International）の主任インストラクターを務め，オブジェクト指向テクノロジーおよびC++のセミナーコースの設計と講義を行っている．

C++ ＆ OOP 用語集
仮想基本クラス　間接的に基本クラスを派生クラスに2回以上渡すとき，キーワードvirtualを基本クラス指定子に追加する．

プロトタイプの拡張は不可能

システムが完成して，もちろんバグはありましたが，このプロトタイプをデモンストレーションしたところ，ユーザーはとても喜んで，これは実際の訓練に使えると賛辞を寄せてくれました．また，それと同時に非常に長いリストが出てきました．つまり，新しいアイデア，変更要求，いろいろな注意事項，追加要求，というようなものです．もちろん，私たちはこのような意見をもらうためにわざわざプロトタイプを作ったわけですからとてもハッピーだったわけですが，そういう意見を全部加えて新しいシステムを構築しようとしたところ，これはもうほとんど不可能だということに気づいたのです．

プロトタイプは，そのシステム全体のサブセットというかたちをもっていました．そこで，意見のリストをそれぞれ担当エンジニアに分けて，実際にこのプロトタイプに変更を加えようとしたところ，自分の担当分野に戻ったエンジニアが，これはもう絶対にこういう変更は無理だと，もう全部最初から書きなおさないといけない，おそらく何か追加したり変更したりするよりも，書きなおしたほうがずっと楽だということに気がついたわけです．

こうしたわけで私たちは，それまで非常に長い時間をかけて作り上げてきたものを全部捨てなければいけなくなってしまったのです．

プロトタイプに追加変更を加えるよりも，最初からやりなおしたほうがずっとやさしいということはわかっていたのですが，それでも，今までやってきたことが無駄になってしまったということに対する不満は非常に強いものでした．せっかく作ってきたものですから，プログラマにも愛着があるわけです．また，ビジネスという面から考えても，15人年以上も投資されていたものを捨ててしまわなければならないのですから，無駄としかいいようがありません．

オブジェクト指向テクノロジーの導入

このとき私たちは，もう一度新しいシステムを作ったとしてもまた同じことが起こるだろうと予想できました．ユーザー達は，新しいシステムを使ってみて，それに対する変更，追加要求を当然出してくるだろうと思ったのです．そのため，システムを1から作りなおさなければいけなくなるような事態を防止する方法を探しはじめました．

私たちは，システムを非常に柔軟性のあるもの，フレキシビリティの高いものにしたい，しかも拡張性をもっていて，将来いろいろな点が変更できるようにしたいと考えました．そこで，プログラム設計にオブジェクト指向テクノロジーを導入し，実際のプログラミング言語としてはC++を使うということに決定したわけです．

C++を採用した理由は，オブジェクト指向型のほかの言語に比べてオーバーヘッドが少なく，効率も非常に高い，実際に使える言語であるといわれていたからです．これが1987年で，ちょうどC++が発表されて1年目くらい，まだC++を使っている人も少なく，C++も初期の段階でした．また，このときはまだオブジェクト指向データベースというものが存在しなかったので，引き続いてオラクルを使うということになりました．

オブジェクト指向導入の結果

オブジェクト指向テクノロジーを導入したことは，結論としてはうまくいきました．むしろ，私たちが思っていたよりも早く成果が出てきたので驚いたくらいです．

初期の段階では，まず，コンピュータサイエンスをまったく知らないユーザーとのコミュニケーションがとてもスムーズに行えるようになったということがあげられます．オブジェクトまたはメッセージといったオブジェクト指向テクノロジーの概念は，ユーザーたちにとっても理解しやすいものだったのです．分析の段階から，このコミュニケーションに助けられて，システムの開発がとてもやりやすくなりました．

もちろん，問題もありました．プログラミングの考え方をオブジェクト指向型に切り替えるわけですから，オブジェクト指向についての勉強が必要ですし，実際にオブジェクト指向テクノロジーを効果的に効率よく使えるようになるまでには，かなりの時間がかかりました．

とくに手続き型に慣れたプログラマであればあるほど，発想の転換をするのに長い時間がかかるというのが初期の段階の傾向としてあります．実際，今までとまったく違った考え方，まったく違った言語，ツールを使うということになるわけです．

また，このような開発チームのメン

C++ & OOP 用語集
可視性　結び付いているオブジェクトに正当にアクセスできるソースコード区域を識別子の可視性があるという．

バーは，自分はすべてを知っていると思っているエンジニアばかりなわけです．C言語も十分に使いこなし，データベースのことも全部わかっていると……．ところが，突然，自分がまったくの新参者になったかのように，まったくわからない世界に入ってしまうのですから，初期の段階でシステムエンジニアが受けるフラストレーションは相当なものです．実際，「以前のほうがもっと簡単だったじゃないか」「なぜオブジェクト指向でやらなければならないんだ」という声もずいぶんありました．

しかし，学習の段階を乗り越えたあとでは，すべてのシステムエンジニアがオブジェクト指向テクノロジーを採用してよかったと喜んでくれました．

オブジェクト指向テクノロジーの発展

新しい考え方ではない！

まず，オブジェクト指向の考え方というのは新しいものではありません．最初のオブジェクト指向プログラミング言語であるSimulaは，1967年に発表されています．このときすでに，オブジェクトという言葉を用いています．次が，Smalltalkでした．Smalltalkの最初のバージョンは1972年に発表されています．

実際，コンピュータサイエンスのいろいろな分野における研究で，オブジェクトというもの，概念としてのオブジェクトというものはずいぶん使われてきています．フレームであるとか，あるいはデータ間の関係性というように呼び方はいろいろですが，オブジェクトについては，かなり前から研究の対象となっています．人工知能の領域においても，知識というものをどのように表現するかということについては，これまでにもずいぶん研究されてきています．

このように，さまざまな分野で行われてきた研究が統合されたかたちで現れたのが，オブジェクト指向モデルだといえます．

ソフトウェアの再利用性

最初に出てきた真の意味でのオブジェクト指向プログラミング言語はSmalltalkですが，Smalltalkが登場してきた目的というのは，プログラマに対して非常にいいツール，生産性が絶対にあがるようなツールを提供するということでした．Smalltalkの開発者たちの基本的な考え方というのは，開発にあたっていちばん高価なものはプログラマの時間であるというものです．とくに，彼らが変えたいと考えていたのは，手続き型プログラミングにおける，プログラマが新しくプログラミングをする場合には，つねに0からもう一度スタートしなければいけないという状況でした．それまでは，ソフトウェアには再利用性がまったくなく，プログラマはどんなにシンプルなものであっても，全部最初から作りなおさなければならなかったのです．

2番目の問題としては，ソフトウェアが1回書かれると，戦略ゲームシミュレーションシステムの例のように，これを変更したり拡張することが非常に難しいという点があげられます．

結論として，手続き型のプログラムでは，一部分になにか変更を加えたい，または追加をしたいという場合には，どうしてもシステム全体に影響を与えてしまうことになる．システム全体に影響を与えず，一部分だけを切り放してその部分だけに変更を加えるということが非常に困難なのです．

おそらく，オブジェクト指向プログラミング言語を開発した人たちは，システムを分割してプログラミングするいい方法がないかと考えたのだと思います．

オブジェクト指向的なアプローチ

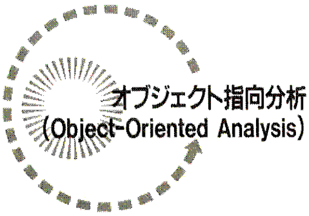
分析の段階からスタートする

まず，とても重要な点として強調しておきたいのですが，オブジェクト指向アプローチというのは，プログラミングの段階からスタートするのではなく，実際には分析の段階，少なくとも設計の段階からスタートするのです．したがって，オブジェクト指向アプローチの場合には，すでに分析の段階から手続き型のアプローチとは違うのだということを明確にしておきたいと思います．

オブジェクト指向アプローチにおいては，オブジェクト，継承といった概念を，分析，設計，プログラミングの各フェーズで一貫して使用します．オブジェクト指向分析で得たオブジェクトのセットを用いてオブジェクト指向設計を行い，オブジェクト指向設計で得た問題解決をガイドラインとして実際のプログラミングを行うというわけ

C++ & OOP 用語集
型宣言　各種変数を利用するときコンパイラに対して宣言をする．宣言しなくても自動的に型を判断して変数を用意してくれる処理系もある．

です．



一般に分析というのは，問題は何かということを調査する，明確にする段階です．

オブジェクト指向アプローチでは，現実の問題領域にあるオブジェクトまたはクラスというものを明確にしていくことになります．

ひとことでいうと，問題領域でのアクター，つまり役割を演じているオブジェクトが何であるのかを探すのがオブジェクト指向分析です．つまり，オブジェクト指向分析では，その問題領域で実際にアクティブに動いているもの，オブジェクトがどれかということを見つけようとするわけです．たとえば，交通管制システムにおいては，信号機，車，道，歩行者などのオブジェクトを見つけることができるでしょう．

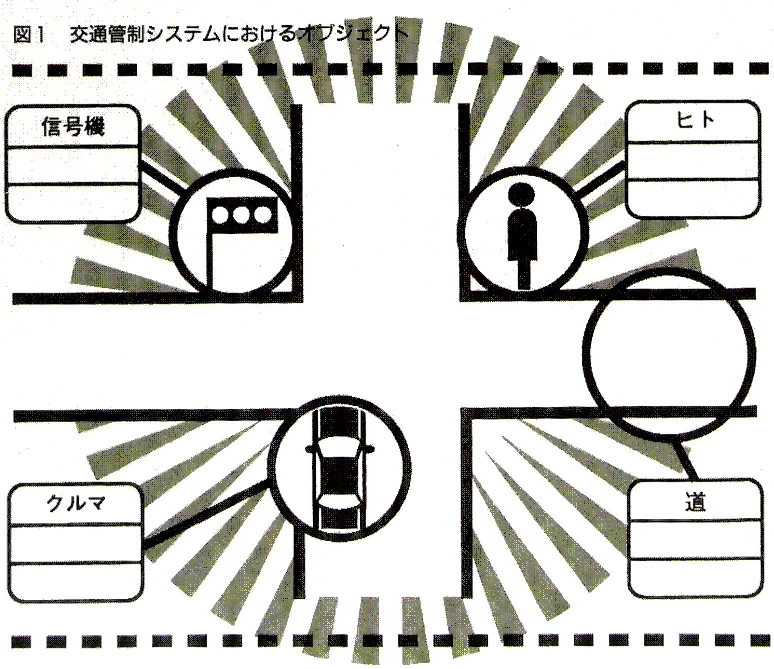
実際のオブジェクト指向分析においては，まずオブジェクトを確定し，そのオブジェクトが記憶していなければいけないものを決め，さらにそのオブジェクトの機能を考えます．同時に，これらのオブジェクト間の関連性，相互作用についても明確にします．つまり，そのシステムの要求がなにかということを明確にするのです．



分析が終ると設計段階に入るわけですけれども，このときはすでに，問題領域の中でなにか役割を果たしているオブジェクトのセット，つまり，そのシステムの要求が明確になっています．そこで，その問題解決をどうするか，インプリメンテーションをどうするかというのが設計段階の課題になります．

実際の設計において，手続き型とオブジェクト指向型の間には非常に大きな相違点があります．通常手続き型の場合には，システムがなすべきメイン関数というものがまず最初にあり，次にこのメイン関数を補うサブ関数，サブ関数の下にさらに小さなサブ関数というようにして，非常にシンプルなところまでずっと分割していくかたちをとります．

図1　交通管制システムにおけるオブジェクト



オブジェクト指向型の考え方においては，もっと広い範囲を捉えた考え方をします．たとえば，小さなソフトウェアを1つ書く場合にも，1つのアプリケーションだけのために書くのではなく，いろいろなアプリケーションに使えるものとして書くわけです．

次に，その問題の中で何か役割を果たしているアクター（オブジェクト），つまり部品は何かということを考えます．そして問題解決の際には，どのように部品を組み立てて実際に問題解決をするかということを考えるわけです．この設計の段階で問題解決に使用するオブジェクトのセットは，分析段階で見つけたオブジェクトのセットです．それをこんどは明確に識別して，問題解決に使えるものを探し，組み立てるわけです．これは，ハードウェアとかメカニカルなエンジニアリングで考えられているのと同じアプローチの仕方です．たとえば，自動車を作るような場合も，車体，エンジン，車輪など，いろいろな部品があり，それらを全部組み立てて自動車ができるということになります．

また，オブジェクト指向設計でやるべきこととして，分析の段階で発見できなかったオブジェクト（これはインプリメンテーションに必要な抽象的なものであることが多い）の追加，オブジェクト定義の詳細化，再利用性の確認，問題解決の方法を詳細に文書化するなどがあります．

C++ & OOP 用語集
型変換　ある型の値を別の型に変更すること．演算の際に必要となる場合もある．自動的に変換して演算を行う処理系もある．

図2　手続き型の設計

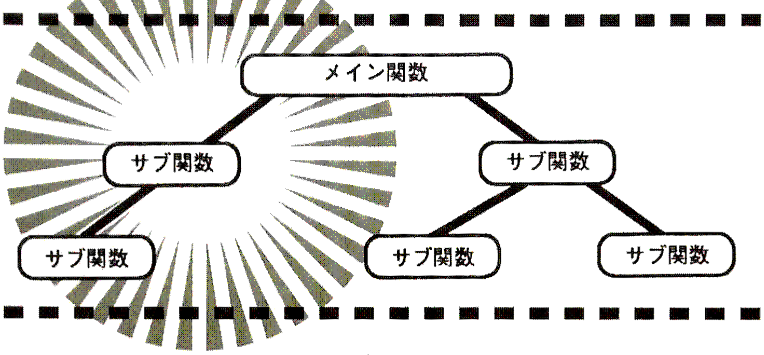
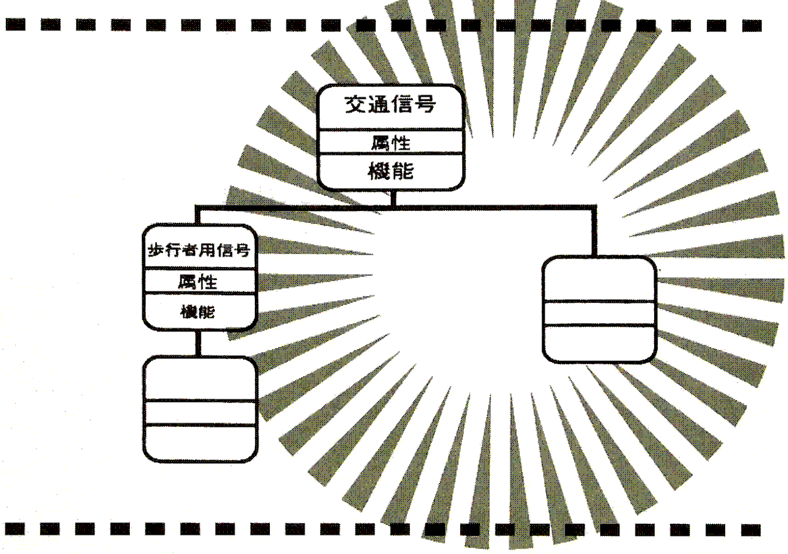


図3　オブジェクト指向設計



インプリメンテーション (Implementation)

インプリメンテーションにおいてもっとも注意すべき点は，既存のものはできる限り利用するということです．たとえば，スタック，リスト，キュー，ハッシュテーブルのような一般的なものは，すでにクラスライブラリの中にクラスとして登録されています．このようなものは，そのまま使ってしまえばいいわけですから，新たにインプリメンテーションせず，そのクラスのオブジェクトを使用すると宣言すればばいいのです．

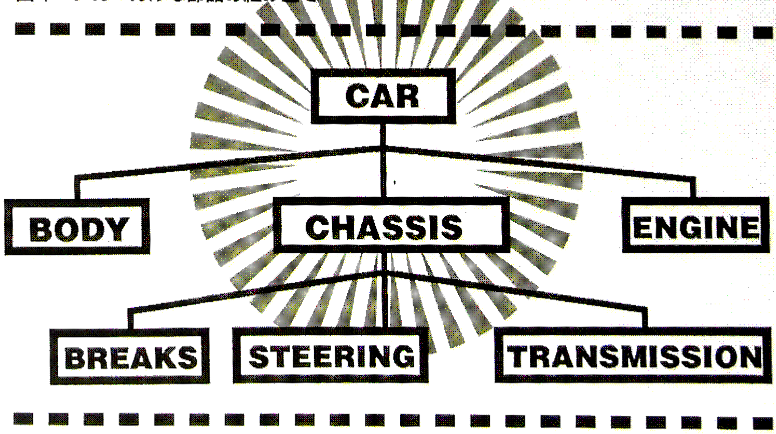
また，新たなクラスを作成する場合には，その再利用性について十分に考慮して行うことが必要です．

現在では，それぞれの問題領域ごとに，多くのクラスライブラリが存在しています．よく利用されるものとして，ユーザーインターフェースを構築するのに必要なクラスをまとめたクラスライブラリなどがあります．おそらく，今後こういう問題領域ごとのクラスライブラリというのは増えていくでしょう．そうなれば，実際のインプリメンテーションは必要なく，市販されているクラスライブラリからそのままとってきて使うだけになると思います．

オブジェクト指向テクノロジーを使っているソフトハウスでは，徐々にそのソフトハウス独自のクラスライブラリが蓄積されていくでしょう．こういうクラスライブラリが豊かであればあるほど，新しいアプリケーションの作成は容易になります．プログラマは，問題解決に合わせてそれらを組み合わせていけばいいわけです．

このようなことから，企業にとってのソフトウェアは再利用可能な資源であるという，新しい考え方が出てきています．クラスというものが，経済的にみても非常に価値の高い資産になるわけです．

図4　メカニカルな部品の組み立て



仮定義　ANSI C規格で導入された新しい概念．記憶クラス指定子と初期化子のないすべての外部データ宣言は，仮定義とみなされる．

オブジェクト指向プログラミング言語

どの言語を用いてプログラミングするか？

オブジェクト指向プログラミング言語には，２つの種類があります．１つはSmalltalkのようなオブジェクトベースの言語，すべてがオブジェクトであるという言語です．もう１つは，Object-CやC++などで，これらはオブジェクト指向モデルをサポートしていますが，すべてがオブジェクトからなりたっているわけではありません．

C++の利点

C++は，C言語をベースにしたプログラミング言語で，オブジェクトやクラスを利用できるC言語といえます．C++は，Cよりももっと抽象的な概念，クラスを扱うことができるのですが，これはC言語の構造体を拡張したものです．

私がオブジェクト指向プログラミング言語としてC++を薦めるのは，オーバーヘッドが少なく，現在のシステムで効率よくプログラムを実行できるからです．つまり，オブジェクト指向テクノロジーを用いながら，しかも効率も捨てなくてすむのです．したがって，C++はプロトタイプや実験だけではなく，実際のプロジェクトに対して有効な言語だといえます．

教育用にはSmalltalkが最適

オブジェクト指向をもっときれいなかたちで行う場合には，Smalltalkなどのピュアなオブジェクト指向プログラミング言語が有効です．実際そういう意味で，オブジェクト指向を学ぶためのツールとしては，Smalltalkのほうがずっといいといえます．なぜなら，Smalltalkにはオブジェクトしかない，オブジェクト，メッセージ，そういう概念しか出てこないわけですから，混同する余地がないのです．

Smalltalkの問題点は，まだ私たちが期待するほど効率がよくない，大型のプロジェクトやプロダクションの場合には，かなりのスピードが要求されますが，残念ながらSmalltalkは追い付けないということです．

1992年，C++は主流言語になる

次に，C++をほかのハイブリッドなオブジェクト指向プログラミング言語と比べた場合，私はC++がベターだとはいいません．プログラミング言語の良し悪しを述べるのは，たいへん危険で難しいことです．ただ，1992年までにプログラミング言語の主流はC++になるだろうといわれていることだけは確かです．

その理由として，C言語を使っているプログラマが非常に多く，CのプログラマがC++にシフトするのは容易であるということがあります．ほかに理由があるとすれば，それは政治的なものだと思います．とにかくはっきりしているのは，将来のプログラミング言語としてC++は主流になるだろうということです．

オブジェクト指向の問題点

身に付けるのは決して容易ではない

オブジェクト指向テクノロジー自体に関しては，とくに問題点はないと思います．ただ，オブジェクト指向テクノロジーを見る人の見方，見ている人のほうにマイナス部分があると思います．

オブジェクト指向テクノロジーは今大きな注目を浴びているものですから，関係する文書もたくさん出ていますし，本も出版されています．ありとあらゆるものにオブジェクト指向という言葉が氾濫しています．

オブジェクト指向が流行になり，誰も彼もが自分がオブジェクト指向だといいたがっているわけです．ソフトウェアベンダーも，なんでもかんでもオブジェクト指向だといって，オブジェクト指向を宣伝材料にしています．実際どのパンフレットを見ても，必ずオブジェクトという言葉が入っています．オブジェクト指向という言葉を入れれば売れると考えられ，その結果すべてがオブジェクトナントカになってしまったわけです．

しかもその説明には，オブジェクト指向を使えば一瞬のうちにすべての問題が解決するようなことが書かれています．こういう書き方をされるので，

C++ & OOP 用語集
カプセル化　プログラムコードとデータを一体化して，クラスオブジェクトを作ること．

プログラマもなにがしかの期待をするわけです．なにかひとつ記事を読めば，それだけでオブジェクト指向テクノロジーが身について，すぐにオブジェクト指向型の設計やプログラミングができると期待してしまいます．

学習には投資も必要

現実に，ほとんどのプログラマが，オブジェクト，継承，メッセージ，というような言葉を覚えています．ところが，実際にそれがどういうことを意味しているのか，どこでどのように使われるのかということについては，ほとんど理解していません．それにもかかわらず，一般の人々はこういう記事を読んで，これでもうオブジェクト指向は全部わかった，これでC++を使ってすぐにもプログラミングができるなと錯覚してしまうわけです．

誰もがオブジェクト指向はすごく自然で，すごく簡単だといっているものですから，ほとんどの人が近道をしてしまいます．C++を使ってすぐにプログラミングをはじめようとか，すぐさまオブジェクト指向データベースを使おうということになってしまう．

しかし，近道をしてしまったために，基本的な概念，基本的な知識というのを押さえていない．ところが，実際にオブジェクト指向の優れたシステムを作り上げるためには，基本的な知識をよく理解していることが重要になるわけです．知識が十分でないと，いろいろな問題が出てきたときに，それを克服できません．

このような風潮が広まってしまったため，どうしても基本的な概念，コンセプト，実際のプログラミング，設計，基本的な用語をよく理解するために，あまり投資をしようとしないのです．これが，オブジェクト指向のひとつの問題点だと思います．

オブジェクト指向のエキスパートになるためには，かなりの時間を学習に費やすことがどうしても必要になります．パイロットプロジェクトと呼ばれるような，小さなプロジェクトを実際に経験し，いろいろなミスを犯してそれを克服するという，試行錯誤を繰り返しながら学習をしていくことが大切です．そこで，オブジェクトを使用した設計の仕方や，オブジェクトをどのように使うのかということを，徹底して学ぶことが必要なのです．

オブジェクト指向プログラミング文献リスト

オブジェクト指向入門
Bertrand Meyer著，二木厚吉監訳，（株）アスキー

オブジェクト指向のプログラミングーソフトウェア開発技法の進化ー
B.J.コックス著，前川守監訳，（株）トッパン

プログラミング言語C++
B.Stroustrup著，斎藤信男訳，（株）トッパン

Object-Oriented Design with Applications
Grady Booch, The Benjamin/Wimmings Publishing Company, Inc. 1991.

Object-Oriented Analysis,
Peter Coad & Edward Yourdon, Prentice Hall, 1990.

Object-Oriented Concepts, Database and Applications
Won Kim & Fred Lochovsky (editors), ACM Press, 1989.

Object-Oriented Programming and C++
Richard S. Weiner & Lewis J.Pinson, Addison Wesley Publishing, 1988.

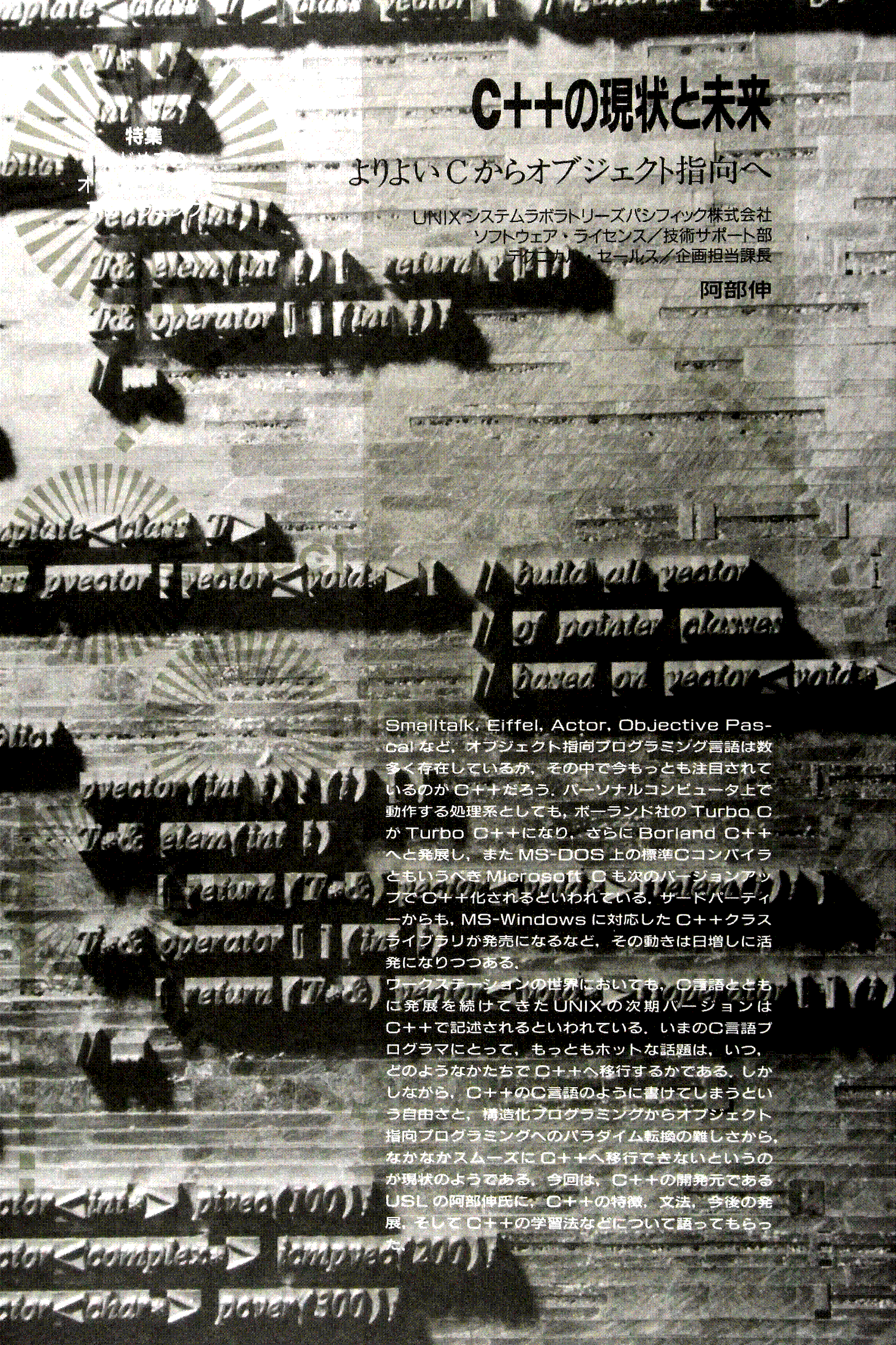
Object-Oriented Program Design with Examples in C++
Mark Mullin, Addison Wesley Publishing, 1989.

C++ Primer
Stanley B.Lippman, Addison Wesley Publishing, 1989.

The C++ Workbook
Richard S. Weiner & Lewis J.Pinson, Addison Wesley Publishing, 1990.

Smalltalk-80: The Language and its Implementation
Adele Goldberg & David Robson, Addison Wesley, 1983 & 1989.

C++ & OOP 用語集
記憶クラス　変数をメモリエリアに保持する期間や，結合状態を示す属性．宣言の構文，またはその構文が置かれた位置によって確立される．



特集

C++の現状と未来

よりよいCからオブジェクト指向へ

UNIXシステムラボラトリーズパシフィック株式会社
ソフトウェア・ライセンス／技術サポート部
テクニカル・セールス／企画担当課長

阿部伸

Smalltalk, Eiffel, Actor, Objective Pascalなど，オブジェクト指向プログラミング言語は数多く存在しているが，その中で今もっとも注目されているのがC++だろう．パーソナルコンピュータ上で動作する処理系としても，ボーランド社のTurbo CがTurbo C++になり，さらにBorland C++へと発展し，またMS-DOS上の標準CコンパイラともいうべきMicrosoft Cも次のバージョンアップでC++化されるといわれている．サードパーティーからも，MS-Windowsに対応したC++クラスライブラリが発売になるなど，その動きは日増しに活発になりつつある．

ワークステーションの世界においても，C言語とともに発展を続けてきたUNIXの次期バージョンはC++で記述されるといわれている．いまのC言語プログラマにとって，もっともホットな話題は，いつ，どのようなかたちでC++へ移行するかである．しかしながら，C++のC言語のように書けてしまうという自由さと，構造化プログラミングからオブジェクト指向プログラミングへのパラダイム転換の難しさから，なかなかスムーズにC++へ移行できないというのが現状のようである．今回は，C++の開発元であるUSLの阿部伸氏に，C++の特徴，文法，今後の発展，そしてC++の学習法などについて語ってもらった．

C++の歴史

ビジャーン・ストラウストラップ（Bjarne Stroustrup）がC++を作ったわけですが，その発端となったのは，彼がケンブリッジの学生だったときに行っていた研究です．このとき，彼は何かのシミュレーションをするような研究課題に取り組んでいたのですが，その実現に最初はSimula67を使用したわけです．

ご存じのように，Simula67というのはデータとその操作をカプセル化する抽象データ型であるクラスの概念を初めて導入した言語で，その点からいえばSmalltalkの親にあたります．ところが，Simula67の処理系がインタプリタであったことと，当時のハードウェア環境がそれほどよくなかったために，ビジャーンがSimula67で作成したプログラムは非常に遅くて，使いものにならなかったということです．

ただ，Simula67のカプセル化機能のおかげで効率よく研究の内容を詰めることができたといいます．最終的なインプリメンテーションに関しては，BCPLという，これはCの親にあたる言語ですが，これを使用して実際に見合う結果が得られたわけです．



写真1 阿部伸氏

ビジャーンは，もしもこの研究で最初からBCPLを使用していたとしたら，とても考えがまとまらずに，最終的なプログラムまで行き着けなかっただろうといっています．

このような経緯を経て，ビジャーンは，Simulaのエレガンスと BCPLの性能を兼ね備えたよりよいプログラミング言語が作れないだろうかと考えはじめたわけです．そして1984年に，彼が最初に出したプログラミング言語が「クラスをもったC（C with Class）」です．この言語は，ただ強化されたタイプチェック機能とクラスをもっているというだけで，仮想関数やオーバーローディングなどの概念も含んでいないたいへんプリミティブなものでした．

1985年に最初にC++トランスレータとしてAT&Tが製品化したのが，この「クラスをもったC」にオーバーローディングなどを付け加えたものです．このあと，protectedクラス・メンバを導入したのが，1986年のリリース1.1です．さらに，マイナーな改良が行われたリリース1.2が1987年に出ています．

そして，マルチインヘリタンス（多重継承）を導入したリリース2.0が1989年に出ています．ここで，基本的にC++が必要とする機能をすべて提供したといっていいと思います．実際，このリリース2.0以降，いろいろな方面におけるC++の需要が増え，世間に広まるようになりました．

現在のC++リリース2.1というのは，リリース2.0の完成度をさらに高める製品として1990年の春にリリースされたものです．そして，今，次のリリース3.0を出すための準備を行っているわけです．

C++の位置付け

UNIXシステムラボラトリーズ（USL）としては，C++をオブジェクト指向プログラミングのための言語という以上に実用的開発言語であると位置付けています．

重要なことのひとつは，C++はオブジェクト指向プログラミング言語の側面と従来の手続き型プログラミング言語であるCとしての側面の両方をもち合わせた言語だということです．このために，純粋なオブジェクト指向プログラミング言語でないという批判もあるようですが，その柔軟性のため適用範囲も広く極めて現実的な開発用言語といえるでしょう．

C++の位置付けを，オブジェクト指向言語におくのか，それともよりよいCにおくのか，ということですが，これはUSLがC++をどういう形で押すかということよりも，C++がどのようにプログラマに受け入れられるのかという問題だと思います．基本的には，プログラマの使いやすいように受け入れてもらえれば，それでよいと考えています．

実際，AT&Tの内部で使った例で，今までの構造化プログラミングとC++を使用したオブジェクト指向プログラミングの間にはっきりと貢献度の違いが出ています．こうした内部的に評価した結果を通して，C++が有用であるということを見極めています．

C++ & OOP 用語集
記憶クラス指定子 auto　ローカルスコープ変数宣言においてのみ使用される．ローカルスコープ変数宣言のデフォルトのため，ほとんど使用されることはない．

やはり，実際に使える言語だというのが，C++の売り文句だと思います．

ほかのオブジェクト指向言語との比較

C++の創始者であるビジャーン・ストラウストラップはある公演で，「C++対純粋OOP」という比較をやっていまして，そこで彼はC++の強みとして移植性，適応性，強力なタイプチェック機能，Cとの互換性そしてマルチパラダイムなどをほかのオブジェクト指向プログラミング言語にまさるものとして列挙しています．

これに対して欠点として，開発環境の不備，コンパイル・リンク時間，純粋でないことそしてCとの互換性などを指摘しています．

開発環境については，現在多くのすばらしい製品が発表，開発されているようですので，改善の方向にあるといえます．コンパイル・リンク時間に関しては，最近のハードウェアの高性能低価格化を考えますと，あまり大きな問題ではなくなると思います．また，インクリメンタルコンパイレーションのような技術が将来導入されれば，さらによくなるでしょう．

おもしろいのは，マルチパラダイム性とCとの互換性が長所と短所の両面を有していることを指摘している点です．しかしこれは，問題点を十分に理解して対処すれば，長所のみを利用することもできるということです．

以上に加えて，現在C++はほかのどのオブジェクト指向言語よりも多くのユーザーに利用されています．また，C++で開発されたソフトウェアの量も著しく増え，その質も高いものになってきています．これらの事実は，C++がいかに実用に耐える言語かを示していると思います．

C言語との違い

クラス

いちばん最初に「クラスをもったC」といわれていたように，クラスの導入というのが行われたわけですが，クラスというのは基本的には構造体で，おもな違いというのはその中でデータ隠ぺいの方法としてprivate，protected，publicといったラベリングができる点です．

基本的には，外に出したくないデータをprivateというラベルの下におき，それへのアクセスメソッドとしての関数をpublicの中に定義する．このメソッドがインターフェースの役目をしてブラックボックス化が可能になるというのが，C++の考え方です．

クラスというのは1つの型で，コンストラクタとデストラクタというクラスオブジェクトを初期化・開放する関数がついています．クラスとして定義されると，そのメンバは明示的にpublicとして宣言されない限り，privateとみなされます．つまり，publicと宣言したところだけが外からオープンな部分になるわけです．

オブジェクト指向の重要な機能の1つとして，C++では親となるあるクラス（ベースクラス）から，その子供となるクラス（導出クラス）を“導出”することが可能です．ここでいうベースクラス，導出クラスとは，ほかのオブジェクト指向プログラミング言語で呼ぶスーパークラス，サブクラスにほぼ対応します．

この導出によって新しく定義されるクラスは，親クラスで定義されたものを継承できるので，再定義する必要がなくなります．ただし，継承されたクラスメンバへのアクセスについてはいろいろな制限があり，クラスの独立性，モジュール性などを維持しています．たとえば，導出クラスからはそのベースクラスのprivateメンバには直接アクセスできません．また，protectedメンバは導出チェーン上のクラスからはアクセスできますが，publicメンバのようにクラス外からオブジェクトを通して直接アクセスすることはできません．

これらとは別に，導出に際してベースクラス全体をprivateと指定すると，ベースクラスのpublicメンバは導出クラスのprivateメンバと同等に扱われ，外から直接アクセスできなくなり，以降の導出クラスからも直接アクセスできなくなります．

強い型チェック

強い型チェックについてですが，リスト1の例1をCとC++でコンパイルしてみます．ここでsqrt()はmath.hというヘッダファイルを必要とするので，C++はその旨の警告を

C++ & OOP 用語集
記憶クラス指定子extern　ファイルスコープあるいはローカルスコープをもつ関数および変数の宣言において外部リンケージを示すのに使われる．

出しますが，Cではそのままコンパイルされてしまいます．

そこで，math.hを加えて例2のようにして，もう一度コンパイルしてみます．こんどはどちらも正常にコンパイルできますが，これを実行するとCでは正確な値が得られません．なぜなら，sqrt()という引数にdoubleをとる関数に整数の2を入れているからです．しかし，C++はsqrt()がdoubleをとるということを解釈していますので，キャストをするなり，適当な処置をしながら結果まで導いてくれます．

定数識別子

定数(const)識別子は，初期化後の変更は不可能ですが，変数と同様に働きます．つまり，型チェックの対象になるわけです．Cで，リスト2の例1のように記述していたのが，C++では例2のように記述できます．

インライン関数

インライン関数も非常に便利なもので，今までは#defineマクロの形で定義していたものをインラインで展開できる関数として定義できます（リスト3）．短い関数をすべてインライン関数で書けば，関数呼び出しのプロセスがないわけですから，かなり効率が高くなります．さらに，今までのマクロのような不統一かつ不鮮明なこともなくなるというメリットがあります．もちろん，型チェックなどはきちんとやってくれます．

データ抽象

ここでは，実際にストリングを結合するプログラムをCとC++を使って書いた例をあげて，その違いを示したいと思います（リスト4）．

C++にはオーバーロード（多重定義）という機能があり，この例では“+”と“=”がそれぞれ定義し直されています．普通ですと，“+”はストリングタイプの加算は定義されていないのでできません．しかしC++の場合は，新しいタイプ（クラス）ごとにオペレータを再定義することで，今までの定義に重ねていくことができるわけです．

C++はオペランドの型をみて，そのオペランドに適したオペレーションを呼び出します．

継承

リスト5は，継承の例を示しています．まず，class Vehicleというのがメインのクラスとして定義されています．次に下位のクラスとして，CarとTruckというのを定義しているわけです．class Carのあとに“: public Vehicle”として，Vehicleというベースクラスの導出クラスであるということを定義しているわけです．

リスト1　強い型チェック

例1

```
#include     <stdio.h>

main()
{
     printf("%¥n", sqrt(2));
}
```

例2

```
#include     <stdio.h>
/* try #including math.h */
#include     <math.h>

main()
{
     printf("%¥n", sqrt(2));
}
```

リスト2　定数識別子

例1

```
#define MAXCHARS 80
```

例2

```
const int MAXCHARS = 80;
```


C++ & OOP 用語集
記憶クラス指定子 register　ローカル変数宣言と関数パラメータ宣言にのみ指定することができる．変数をレジスタに割り当てることにより効率を高めることが可能．

publicと宣言することで，ベースクラスのpublicメンバはclass Carのpublicメンバと同様にアクセス可能になります．これをprivateと宣言すると，これ以降，たとえばclass Carの下位クラスで，Vehicleのクラスメンバにまったくアクセスできなくなってしまいます．

ダイナミックバインディング

リスト6は，ダイナミックバインディングの例です．まずShapeというベースクラスがあって，その下にRectangleとCircleという2つのサブクラスがあります．定義の仕方はまったく同じですが，違いはいちばん上のベースクラスの中にVirtualという言葉が出てくるというところです．このVirtualという定義が，C++におけるダイナミックバインディングを可能にしているわけです．

実際にdraw()を呼び出すのに，たとえばRectangleクラスのオブジェクトAというのを定義すれば，A.draw()とすればこれが呼び出せるわけです．このときは，もうA．ではっきり型が決められてしまっています．しかし，たとえばShapeクラスのポインタを定義して，そのポインタがRectangleを指すかもしれないし，Circleを指すかもしれないし，またはTriangleを指すかもしれない．そういう状況で，そのShapeポインタでdraw()を呼び出しているというような場合（Pがそういったポインタとすると，P→draw()）．draw()というのが，どのクラスに対応したdraw()なのか，プログラミングの段階ではわかりません．しかし，ここでvirtualと宣言することで，適当なクラスのdraw()を呼び出すことができるようになります．

リスト3　インライン関数

例1：C言語による記述

```
#define max(x,y) ¥
    ((x) > (y) ? (x) : (y))
```

例2：C++による記述

```
inline  int max(int x, int y)
{
    return (x > y ? x : y);
}
```

リスト4　データ抽象

例1：C言語による記述

```
char    *machine, *service;
char    *dialstring;

dialstring = malloc(strlen(machine)
    + strlen(service) + 2);

if(dialstring == NULL)
    error();

strcpy(dialstring, machine);
strcat(dialstring, ".");
strcat(dialstring, service);

/* ... */
free(dialstring);
```

例2：C++による記述

```
stringクラスが正しく定義されているとする

string machine, service;
string dialstring;

dialstring = machine + "." + service;
```

リスト5　継承

```
class Vehicle
{
    int mpg;
    // ...
public:
    void refuel(float);
};

class Car : public Vehicle
{
    short n_passengers;
    // ...
};

class Truck : public Vehicle
{
    long payload;
    // ...
};
```


記憶クラス指定子static　ファイルスコープおよびローカルスコープをもつ関数と変数宣言において，内部リンケージを示すために使われる．

多重継承

多重継承は，複数の親クラスを“：”に続いてコンマでくぎって記述すればよいだけです．あとは普通の継承と同じだと考えてよいと思います．

ただ問題となるのは，1つのスーパークラスがあって，それからAとBが導出されるわけですが，さらにサブクラスCがAとBの両方から導出されると，別の経路を経て同じメンバがだぶって継承されてしまう可能性がでてきます．なにもいわずにこのまま定義してしまうとそうなってしまいますので，スーパークラスをvirtualとして定義することが可能になっています．こうすることで，メンバがだぶるのを避けることができます．

関数宣言における問題

ANSI Cの定義に強い影響を与えてきたことからもわかるように，C++というのはオブジェクト指向のプログラミング言語という見方以外に型チェックのたいへん強いCという見方もできるわけです．

この強いタイプチェックを実現するために，C++では，内部での関数名として関数名自体の中に引数の型まで埋め込んでしまいます．これによって起こる問題のひとつが，Cとのリンクです．このような形の関数名は，まったくCからは読めません．ところが，C++の中からCの標準のライブラリを呼び出すこともあるわけです．

この問題を解決するために，現在のバージョンでは，Cの関数を呼ぶ場合は明示的にこれはCの関数を呼んでいると宣言できるようにしています．こうすることで，C++の関数名の生成をしないようにしているわけです．

オブジェクト指向かよりよいCか？

C++をオブジェクト指向言語として使うのか，あるいはよりよいCとし

リスト6　ダイナミックバインディング

例1

```
class Shape
{
public:
    virtual void draw();
    // ...
};

class Rectangle : public Shape
{
public:
    virtual void draw();
    // ...
};

class Circle : public Shape
{
public:
    virtual void draw();
    // ...
};

// And so on ...
```

例2：draw( )を含む関数

```
void refresh(Shape &shape)
{
    // ...
    shape draw()
    // ...
}
```

例3：C言語による記述

```
void refresh(Shape &shape)
{
    /* ... */
    switch(shape.tag) {
    case rectangle:
        draw_rectangle(shape);
        break;
    case circle:
        draw_circle(shape);
        break;
    /* give cases for all other shapes ... */
    default:
        draw_shape(shape);
        break;
    }
    /* ... */
}
```

C++ & OOP 用語集
記憶クラス指定子 typedef　ほかの記憶クラス指定子とは違いオブジェクトを宣言するのではなく，新たなデータ型指定子を定義するのに使われる．

てだけ使うのかということになりますと，C++は，ANSI Cよりもさらに優れたCという見方で使用することも可能なわけです．そういう使い方をする限りにおいては，分析は構造化分析を行い，構造化プログラミングの手法でデザインし，それでプログラミングしてなんの問題もありません．

ただし，オブジェクト指向としてC++を使う場合には，やはりOOA（オブジェクト指向分析）ですとかOOD（オブジェクト指向デザイン）などの考え方を導入しなければなりません．実際，そのためのツールというものも，現在いろいろなソフトウェアベンダーで作られているようです．

オブジェクト指向というのは，Smalltalkだとか C++だとか，そんなことは関係ないわけです．極端なことをいえば，FORTRANでオブジェクト指向プログラミングができないかといったら，できるわけです．つまり，これは完全に方法論の問題なわけです．

プログラミングをオブジェクト指向で行うと決めたならば，どうしてもオブジェクトの認識とかクラスの定義という，今までの構造化プログラミングとはまったく別のパラダイムのもとにプログラムを作るわけで，OOAやOODを除いて考えるわけにはいかなくなります．

C++をいかに学ぶか

現在，今まで構造化プログラミングを行ってきたプログラマに，OOA，OODを含めてオブジェクト指向プログラミングをどのように教育していけばいいかという大きな問題があります．実際，かなり大きなパラダイムチェンジがないと，根本的なオブジェクト指向プログラミングへ移ることはできないのですが，そのための優れたトレーニングコースはあまり見あたりません．

とにかく1年くらいの間，C++のようなオブジェクト指向言語を使って，オブジェクト指向に親しむべく，実際にプログラミングをやってみるというのが，唯一のよい方法だろうと，アンディ・コーニックもいっています．

やはり，完全にオブジェクト指向プログラミングに移行するにはかなりの時間がかかると思います．しかし，C++の場合は，考え方の切り替えには時間がかかりますが，実際のプログラミングは明日からでもできます．これは，よりよいCとしても使えるC++の大きなメリットです．

はじめはC++を型チェックの強いCとして使い，それからクラスという概念を徐々に導入していく，次に継承の概念を導入し，さらに仮想関数だとかダイナミックバインディングなどをどんどん取り入れていって，最終的にオブジェクト指向のプログラミング手法を自分なりに獲得していくという，このような方法も考えられると思います．

また，C++とかSmalltalkとかいう以前に，オブジェクト指向というプログラミング方法自体が，オブジェクトという非常に人間の意味世界に近いものを認識してやっていくわけで，うまく有効かつ実用的なクラスを定義しづらい場合も多いでしょう．

ですから，最初はかなり限定された世界，たとえば，よくオブジェクト指向に向いているといわれるウインドウだとかGUIのように，限定されていてはっきり定義しやすい世界からやっていくほうが成功するかもしれません．

また，根本的に手続き型の問題をわざわざオブジェクト指向で考える必要があるかどうかという問題があります．すべての問題解決に対して有用なプログラミング言語というのはないんです．たとえば，COBOLならCOBOLの得意とするところがあり，FORTRANならFORTRANの得意とするところがあります．当然，オブジェクト指向言語が得意とする部分というものも出てこざるを得ないと思います．現実問題としても，現在山のように存在しているCOBOLのプログラムを，オブジェクト指向がいいからといって書き換えるかといったら，そんなことはありえません．

C++の場合，Smalltalkのような純粋なオブジェクト指向言語ではありませんから，半分手続き型のプログラミングをして，残りの半分はオブジェクト指向でプログラミングをするということが簡単にできます．こういうことを1つの言語で効率的にできるというのは，大きな利点だと思います．大きなソフトウェアの場合，ある部分，たとえばインターフェースの部分にはオブジェクト指向プログラミングが適当であるかもしれないし，またある部分は根本的に手続き型の世界で，そこをオブジェクト指向型にする必要性さえ感じないかもしれない．このようなときに，ピュアなオブジェクト指向言語でプログラムを作るのはあまり効率がよくないかもしれません．そういう問題に対しては，やはりC++がいちばん強いといえます．

C++ & OOP 用語集
疑似命令　コンパイル時などに，オブジェクトファイルへ変換されるのではなく，処理系に対して指令を行う命令．

C++関連書籍

C++の参考書といいますと，まずビジャーン・ストラウストラップが最初に出した『プログラミング言語C++』(Bjarne Stroustrup著，斎藤信男訳)．これは，いちばん最初のC++のテキストなんですが，残念ながら文章そのものに若干難解な部分があります．これは，ストラウストラップ自身がネイティブなアメリカ人ではないということもありまして，彼自身もかなり苦労して書いたという話を聞いております．そういうわけで，若干読みづらい本になってしまった感があります．内容としては，おもにリリース2.0以前のリリース1.2までのものです．このため，マルチプルインヘリタンス（多重継承）などについての解説は含まれていません．

次に，やはりUSLの人間であるスタンリー・リップマンが『C++ Primer』(Lippman,S.B.著)という名前で本を出しています．この本は，米国ではベストセラーになりました．初心者向けの本としてわかりやすく書かれていて，『プログラミング言語C++』よりも読みやすくなっていると思います．

これはリリース2.0に対応した最初の本であるほか，パラメタライズドタイプやエクセプションハンドリングの記述も多少含まれています．

もう1冊，『The Annotated C++ Reference Manual』(Margaret A. Ellis, B.Stroustrup著）をあげておきたいと思います．これは，マーガレット・エリスとビジャーンの共著です．マーガレット・エリスもUSLのエンジニアで，最近まではおもにサポートなどの仕事をしていました．この本は，ANSIのC++の標準化の作業が開始されるのと同期して，C++の基本的な言語仕様を表すスタンダードな本として，ANSIのベースとなるドキュメントとして書かれたものです．ですから，今いちばんC++がどのような機能を含んでいるかということを正確に表しているのがこの本だと考えていただいていいと思います．これにも，リリース2.0以降に導入される可能性のあるパラメタライズドタイプですとか，あるいはエクセプションハンドリングなどについても記載されています．

USL C++の現状

現在のUSLのC++処理系というのは，C言語へのトランスレータです．いちばん最新のものがリリース2.1というバージョンで，このあとリリース3.0というバージョンが，たぶん今年の第3四半期の後半に出てくると思います．これは，2.1または2.0にパラメタライズドタイプを加えたものです．

さらにリリース3.0以降，ネイティブコードジェネレータやエクセプションハンドリングほかを含むリリース4.0などの計画があるようです．

C++リリース3.0以降で付加される機能

リリース3.0で付け加えられる機能は，パラメタライズドタイプ（parameterized types）と呼ばれるものです．この機能は別名テンプレート（templates）とも呼ばれています．

クラス階層の中で，たとえば配列を作ったとします．すると，intをベースとした配列などがオーバーロードされていくわけですが，そのそれぞれについて，intに対する配列を定義するとか，floatに対する配列を定義するとか，あるいは関数の定義というのを個別にやらないといけません．

ところが，関数それぞれの定義のスケルトンというのはものすごく似ているわけです．たとえば，2つの引数の最小値を求めるような関数というのは，単純に2つの引数を比較してその結果を返しているだけです．その対象とな



写真2　『プログラミング言語C++』
ISBN 4-8101-8002-6 C3055

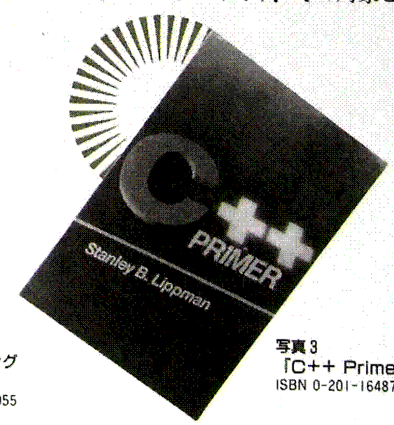


写真3　『C++ Primer』
ISBN 0-201-16487-6

C++ & OOP 用語集
共用体　複数のメンバが集合したデータ構造で，共用体の各メンバが同じアドレスを参照する．

る引数がどんな型であれ，また新しく定義されたクラスであれ，やっていることは同じなんです．

今までは，このような関数をそれぞれの型について個別に定義してやらなければいけなかったわけです．こうした繰り返し定義する必要をなくしたのが，パラメタライズドタイプです．

エクセプションハンドリング(Exception Handling)は，リリース3.0以降に入ってくると思います．また，インクリメンタルコンパイルのような機能も将来的には考慮されるでしょう．

クラスライブラリ

クラスライブラリというのが，オブジェクト指向言語の場合にはかなり重要な要素になります．現在USLが提供しているライブラリには，C++ Standard Libraries, Release 2.0, C++ Standard Library Extension, Release 2.0, C++ Object Interface Library, Release 1.1があります．C++ Standard Library Extension, Release 2.0は，標準のiostreamをクライアントサーバコミュニケーションに対応させたipc-stream，ダブルリンクリスト，アソシエイティブアレイなどをまとめたもので，C++ Object Interface Library, Release 1.1は，いわゆるGUI環境を提供するライブラリで，Motifと OPEN LOOKの両方に対応しています．

将来においては，ANSIの中でライブラリの標準化が行われた段階で，ANSIに準拠したUSLのライブラリを提供できると思います．

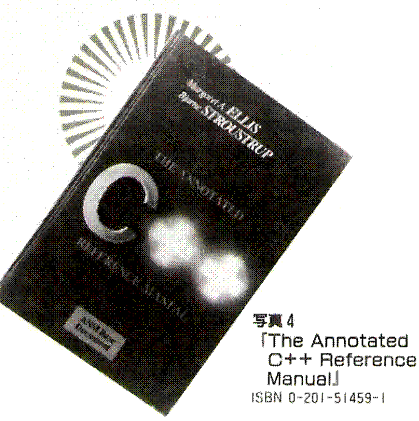


写真4
『The Annotated C++ Reference Manual』
ISBN 0-201-51459-1

USL (UNIX System Laboratories, Inc.)

USLは米国AT&Tの子会社であり，スーパーコンピュータからパソコンまで幅広く利用されているOSであるUNIX System Vオペレーティング・システムをはじめとして，TUXEDOトランザクション処理システム，OSI通信プラットフォーム，C++言語システムなど各種標準に準拠したシステムソフトウェア製品を開発し，全世界のオープンシステムコンピュータ業界に提供している．

これらの製品は，オープンシステムプラットフォーム，ネットワーキング，トランザクション処理，セキュリティ，国際化機能関連製品，システム管理，アプリケーション開発ツールの7つの製品群に分類されている．

USLでは，これらをソースコードライセンスのかたちで提供している．つまりエンドユーザーは，USLから直接製品を購入するのではなく，USLから提供されたソースコードを元にしてオープンシステムコンピュータ業界が開発した製品を購入することになる．

このほかにもUSLでは，企業に対する技術指導や情報提供を行うとともに各種のセミナーなども開催している．

C++ & OOP 用語集
キーワードoperater　演算子を定義するにはキーワードoperaterに続けてその演算子記号をつけたものを名前とする関数を定義する．

COLUMN

C++サンプルプログラム

ここでは，C++の特徴をよく表しているごく短いプログラムを紹介する．これは，アンディ・コーニックが，「An example of dynamic binding in C++」として発表した論文に用いられたプログラムである．

プログラムの内容は，数式をツリー構造を用いて取り扱おうとするものである．図1に示したツリーは，(-5)＊(3+4)という数式を表している．また，リスト1はこのプログラムの使用例で，図2にその出力を示している．

このようなツリー構造を定義するために，クラスnodeとクラスTreeを定義している（リスト2，3）．

クラスNodeのuseというのは，ガベージコレクションのために使う使用回数を記録する変数である．Node()というのはクラスNodeのコンストラクタで，Nodeクラスオブジェクトの生成においては，そのprivateデータメンバであるuseは1に初期化される．

次にクラスTreeをfriend関数として定義している．friendと定義することによって，クラスTreeの中からクラスNodeのprotectedのデータをアクセスすることができるようになる．friend関数はプログラムの中でそれぞれが一応独立していることに意義がある．2つのクラスが参照し合う場合に有効な解決方法として使用される．

“≪”オペレータもfriendになっているが，C++では，2つの引数をもつオペレータの場合，左の引数はそのオペレータの属するクラスを決めるというシンタックスになっているため，“≪”オペレータをNodeクラスのメンバにすることはできないのである．

図1

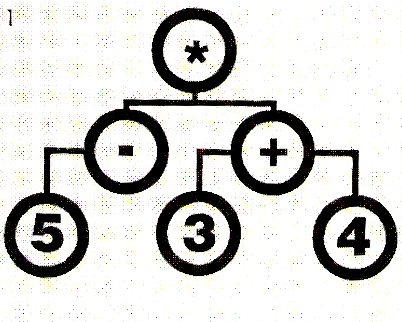


図2

```
((-5)*(3+4))
(((-5)*(3+4)*((-5)*(3+4)))
```

リスト1

a.実際のプログラム例

```
Tree t("*", Tree("-", 5),
     Tree("+", 3, 4));
cout << t << "¥n";
t = Tree("*", t, t);
cout << t << "¥n";
```

b.実行した結果

```
((-5)*(3+4))
(((-5)*(3+4))*((-5)*(3+4)))
```

リスト2　Nodeクラスの定義

```
class Node{
     int use;
protected:
     Node() { use = 1; }
     friend class Tree;
     friend ostream& operator<<
          (ostream&, const Tree&);
     virtual void print(ostream&) const = 0;
     virtual ~Node() {}
};
```

リスト3　Treeクラスの定義

```
class Tree{
     friend class Node;
     friend ostream& operator<<
          (ostream&, const Tree&);
     Node* p;
public:
     Tree(int);
     Tree(char*, Tree);
     Tree(char*, Tree, Tree);
     Tree(const Tree& t) { p = t.p; ++p->use; }
     ~Tree() { if(--p->use == 0) delete p; }
     void operator=(const Tree& t);
};
```


C++ & OOP 用語集
キーワード this　すべての非静的メンバ関数の本体内で使用できるローカル変数のこと．thisは宣言する必要はなく，関数定義内で明示的に参照されることもない．

COLUMN

次にprint()がvirtualとして宣言されているが，これはクラスNodeのサブクラスにIntNode（nodeから枝が1つも出ていないもの，リスト5），UnaryNode（nodeから枝が1つ出ているもの，リスト6），BinaryNode（nodeから2つ枝が出ているもの，リスト7）の3つ存在し，実際にはそれぞれのクラスに対しprint()が対応しなければならないため，ダイナミックバインディングを行う必要があるからだ．

このように，いちばん親になるクラスのところでダイナミックバインディングが必要な関数をvirtualとして宣言しておくことで，プログラム全体をカバーできるようになる．

~Node()というのは，デストラクタで，メモリを解放するための関数である．スコープから抜けたときには，自動的にこのデストラクタが呼び出される．この関数がvirtualとして宣言されているのは，Nodeのサブクラスに種類があり，それぞれデータのサイズが違うので，そのサイズに応じたデストラクタを用意する必要があるからである．

クラスTreeはNodeに対するポインタをもっている．publicとして宣言されているTree(int)，Tree(char＊，Tree)，Tree(char＊，Tree，Tree)の3つは，それぞれIntNode，UnaryNode，BinaryNodeのコンストラクタで，~Tree()はデストラクタである．

アウトプットのオペレータは，クラスTreeとクラスNodeの中でfriendと宣言されている．これは，クラスTreeのプライベートデータメンバであるNodeポインターpとクラスNodeのprotectedメンバ関数print()，これらにアクセスするために必要だからである．

リスト4　出力のオペレータ

```
ostream&
operator<<(ostream& o, const Tree& t)
{
    t.p->print(o);
    return o;
}
```

リスト5　IntNodeの定義

```
class IntNode: public Node{
    friend class Tree;
    int n;
    IntNode(int k): n(k){}
    void print(ostream& o) const
        { o << n; }
};
```

リスト6　UnaryNodeの定義

```
class UnaryNode: public Node{
    friend class Tree;
    char* op;
    Tree opnd;
    UnaryNode(char* a, Tree b):
        op(a), opnd(b){}
    void print(ostream& o) const
        { o << "(" << op << opnd << ")"; }
};
```

リスト7　BinaryNodeの定義

```
class BinaryNode: public Node{
    friend class Tree;
    char* op;
    Tree left;
    Tree right;
    BinaryNode(char* a, Tree b, Tree c):
        op(a), left(b), right(c){}
    void print(ostream& o) const
        { o << "(" << left << op << right << ")"; }
};
```

リスト8　コンストラクタの定義

```
Tree::Tree(int n)
{ p = new IntNode(n); }

Tree::Tree(char* op, Tree t)
{ p = new UnaryNode(op, t); }

Tree::Tree(char* op, Tree left, Tree right)
{ p = new BinaryNode(op, left, right); }
```

リスト9　アサインメントのオペレーション

```
void
Tree::operator=(const Tree& t)
{
    ++t.p->use;
    if(--p->use == 0)
        delete p;
    p = t.p;
}
```

特集
はじめての
オブジェクト指向
プログラミング

Windowsにおける オブジェクト指向とは?

システムとプログラム開発は分けて考える

編集部

Windowsプログラミングは，オブジェクト指向プログラミングであるといわれることがある．Windowsのユーザーインターフェースは，オブジェクト指向であるといういい方をすることもある．オブジェクト指向プログラミングを実際のシステムとして実現したのは，米XEROX社のパロアルト研究所で開発されたSmalltalkシステムであるが，メッセージベースのプログラミング，ビットマップディスプレイを使ってのグラフィック表示，ポインティングデバイスとしてマウスを利用するユーザーインターフェースなどは，MS-Windowsにも大きな影響を与えている．
オブジェクト指向というプログラミングパラダイムはWindowsにおいてどのようなかたちで，どの程度実現されているのだろうか．ここでは，Windowsとオブジェクト指向プログラミングパラダイムの関係について，Smalltalkシステムに代表される典型的なオブジェクト指向プログラミングとの比較を交えながら考察してみたい．また，MS-Windowsのアプリケーション開発環境と，そこで利用できる各種のオブジェクト指向プログラミング言語についてもまとめてみる．

Windowsとオブジェクト指向パラダイム

何が「オブジェクト指向」か?

オブジェクト指向プログラミングに対する関心が高まっている．オブジェクト指向プログラミングを導入すれば，複雑なプログラミングが容易になる，作成したオブジェクトが再利用可能になるため，全体として新しく開発しなければならないプログラムが少なくてすむなど，その有効性が強調される反面，いったい何が「オブジェクト指向」であるかについては，意見が分かれるところも多い．

オブジェクト指向型のプログラミング言語という発想は，構造化，抽象データ型のプログラミング言語の発展したかたちとして，プログラミング言語の基礎研究の分野で生まれた．それが，プログラミングシステムとして統一したかたちで提示されたのはSmalltalkによってであった．現在，オブジェクト指向という言葉を使うとき，それは単にオブジェクト指向型プログラミング言語を指すにとどまらない．データベース設計やプログラム設計・分析，ユーザーインターフェース，人工知能研究など，さまざまな分野に応用されている．

MS-Windowsとオブジェクト指向について考えるとき，Windowsがオブジェクト指向パラダイムに基づいて設計されたシステムであるという問題と，Windowsアプリケーションをオブジェクト指向のプログラミング言語を使って開発するという問題は，独立に取り扱われなければならない．

GUIとオブジェクト指向

複数のウィンドウを画面に開き，マウスを使って操作するグラフィカルユーザーインターフェース（GUI）の開発には，オブジェクト指向プログラミングが非常に有効である．

ウィンドウ，アイコン，スクロールバー，ボタンなどの要素を，それぞれオブジェクトとしてとらえることによって，ビジュアルなオブジェクト（対象）の構造とプログラム上の構造とを明確に対応づけることができるため，対象を自然に記述できる．

たとえば，画面に複数のウィンドウを表示する場合を考えてみよう．それぞれのウィンドウはそれぞれが機能をもち，ユーザーは各ウィンドウごとに個別の操作を行う．各ウィンドウごとの処理は独立していて，ほかのウィンドウに影響を与えてはならないが，どのウィンドウもアイコン化したり，サイズ変更や移動が可能であるといった共通の性質をもっている．つまり，ウィンドウはほかのウィンドウと共通の情報をもっていると同時に，ひとつひとつ個別の性質をもつことになる．これは，オブジェクト指向プログラミングのクラスとインスタンスの概念によく適合する．

米NeXT社のワークステーションNeXT上のGUI環境であるNeXTStep上では，Interface Builderという対話型のアプリケーション開発環境が提供されている．そこでは，ボタンやスクロールバーなどの部品を画面に配置し，その相互の関係を記述したり，あるボタンが押されたときにどのようなメソッドが働くかを記述したりすることが可能だ．具体的でグラフィカルな部品を選択し関連づけ，それをコンパイル，リンクするだけで，ユーザーインターフェースを構築できる．このように，GUIの分野ではオブジェクト指向プログラミングの有効性は非常に高いといえる．

オブジェクト指向とメッセージ

オブジェクト指向プログラミングにおいては，メッセージが唯一のプログラムの表現形式となる．あるオブジェクトの状態を変化させる必要が生じたときに，その必要を生じたほかのオブジェクトが，直接これを変化させるための操作を行うことはできない．オブジェクトは，おのおの情報隠ぺい機能が働いているため，操作できるのはそのオブジェクト自身だけである．外部からオブジェクトに対して操作を要求するには，メッセージを送信する．

オブジェクトに対するメッセージの送出は，手続き型のプログラミング言語における手続き呼び出しや関数呼び出しに相当する．しかし，手続き呼び出しや関数呼び出しでは，呼び出した側の制御のもとにあるのに対して，オブジェクト間のメッセージ送信は，送る側と受ける側が完全に独立しているという点で異なっている．

Windowsはオブジェクト指向か?

C++ & OOP用語集

継承　あるクラスのもっている特性や動きを他のクラスが継ぐこと，またその仕組．もとになるクラスを基本クラス，継承したクラスを派生クラスと呼ぶ．

Windowsは，MS-DOS上でGUIを提供する環境であるため，オブジェクト指向パラダイムときわめて親和性が高い．また，Windowsはメッセージベースのアーキテクチャをもつシステムである．Windowsは，何らかのイベント（キーが押された，マウスが移動したなど）が発生したとき，あるいは何か動作が必要なときにも，必ずメッセージを生成する．そして，アプリケーションにメッセージを送る．Windowsアプリケーションは，Windowsシステムからメッセージを受け取って，速やかにこれを処理し，また次のメッセージを読むという動作を繰り返す．これをプログラマから見ると，各アプリケーションのウィンドウは，メッセージを受け取って処理するオブジェクトであるともいえる．

Windowsアプリケーションの作成時には，どのウィンドウも必ず何かのウィンドウクラスに属していなければならない．同じアプリケーションをもうひとつ起動すると，あとから起動したものも，そのクラスが利用できるようになる．その際，2つのアプリケーションはインスタンスハンドルによって区別される．このクラスとインスタンスの考え方もオブジェクト指向プログラミングのものだ．

しかし，Windowsにおけるこのようなオブジェクト指向プログラミングの要素は，厳密なオブジェクト指向プログラミング環境に比較すると，それほど「オブジェクト指向的」ではない．たとえば，オブジェクト指向型のプログラミング言語であり，かつオブジェクト指向のプログラミング環境をも提供するSmalltalkシステムでは，データやプロセス，アルゴリズム，コンパイラなど，言語処理系として実行する場合に現れるおよそすべての要素を，オブジェクトとしてモデル化しており，オブジェクトの所属（クラス）やオブジェクトの動作，オブジェクトの階層に基づいて厳格に定められている．Windowsプログラミングにおいて，ビジュアルなオブジェクトについてはある程度このような考え方を適用できるが，それ以外の点ではオブジェクト指向型のプログラミングをするかどうかはプログラマの責任である．

Windowsアプリケーション

基本概念はいかに実現されているか？

Windowsアプリケーションにおいて，オブジェクト指向プログラミングの基本概念であるオブジェクト，メッセージ，クラス，インスタンス，インヘリタンスがどのようなかたちで実現されているかをもう少し詳しく見てみることにする．

Windowsにおけるオブジェクト

Windowsの基本であり，かつもっとも重要なオブジェクトはウィンドウである．ウィンドウとは，Windowsアプリケーションのユーザーから見ると，画面上の枠で囲まれた四角い領域である．アプリケーションを起動すると必ずウィンドウが開かれ，処理の内容はウィンドウ内に表示される．ウィンドウは個々のアプリケーションプログラムをほかのアプリケーションプログラムと区別する単位ととらえることもできる．

プログラマから見ると，ウィンドウは単なる画面上の矩形の領域だけを意味しない．各アプリケーションのウィンドウは，ユーザーからの入力やWindowsシステムからの情報をメッセージのかたちで受け取って，これを処理するオブジェクトと考えられる．ここでいうオブジェクトとは，正統的なオブジェクト指向プログラミングでいうところのオブジェクト，データとそのデータを処理するためのメソッドの集合体としてのオブジェクトである．Windows上で動作する個々のアプリケーションプログラムは，それぞれウィンドウを開き，そのウィンドウのスタイル，サイズ，画面上での位置，メニュー，アイコンなどの情報をもっているだけでなく，Windowsシステムから送られているメッセージにどのように対応するかという定義も保持している．メッセージに対応するための定義は，ウィンドウ手続きという関数のかたちで実現されている．どのウィンドウも1つのウィンドウ手続きをもつ．

また，Windowsで定義済みコントロールと呼ばれるボタン，チェックボックス，スクロールバー，エディットボックス，リストボックスなどもオブジェクトとしてとらえることができるだろう．これら，定義済みコントロールは子ウィンドウ，つまり一種のウィンドウとして実現されている．たとえば，ボタンについて見てみると，これは“button”クラスのウィンドウとして，その物理的なサイズやそのボタンが表示されるウィンドウの情報とともに定義される．ボタンが押されたという情報はメッセージのかたちで，そのボタンが置かれているウィンドウに通知されるのである．

C++ & OOP用語集
構造化　プログラム全体の機能を階層的に部分機能に分解し，トップダウンで作成するプログラミング手法．順次，選択，反復だけで表現し，GOTO文は極力使わない．

Windowsのメッセージシステム

Windowsアプリケーションの構造を理解するうえでのキーとなるのが，メッセージループである．ユーザーがあるメニューを選択すると，Windowsシステムはアプリケーションに「メニューが選択された」とメッセージによって通知する．アプリケーションは，Windowsシステムからメッセージを受け取り，その内容に応じた処理を行い，また次のメッセージを受け取って……これを繰り返す．それをメッセージループという．

メッセージループが必要となるのは，Windowsがマルチタスクのシステムであり，複数のアプリケーションが同時に動作する環境だからだ．複数のプログラムが動作するためには，それぞれに対してCPU時間を割り当てなければならない．UNIXのようにはじめからマルチタスクを前提としたオペレーティングシステムでは，あるアプリケーションが処理の途中であっても，ある一定のタイムスライスが経過すると，そのアプリケーションに割り込みをかけてCPUをほかのアプリケーションに渡してしまう．このことをプリエンプティブなマルチタスクであるという．

ところがWindowsは，元来シングルタスクOSであるMS-DOS上で動作するシステムであるため，マルチタスクといってもプリエンプティブではない．Windowsがアプリケーションに対して割り込みをかけることはできない．あるアプリケーションがメッセージを受け取って処理を行い，次のメッセージを調べるため，いったんWindowsシステムに制御がもどる．このときWindowsが，マルチタスク制御システムとしての機能を果たす．

リスト1は，Windowsアプリケーションのメッセージループを示している．メッセージを取り出すGetMessage()関数が呼ばれると，Windowsはそのアプリケーションのメッセージキューをチェックし，メッセージがなければ，同じくGetMessage()関数をコールした状態で待ち続けているほかのアプリケーションのメッセージキューをチェックして，メッセージがあればそのアプリケーションに制御を渡すのである．

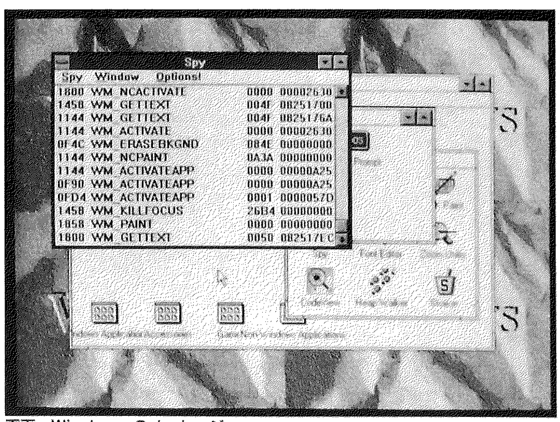
Windowsでは，メッセージはすべて一定の形式をとる．SDKのwindows.hに含まれるメッセージ構造体がそれだ（リスト2）．構造体のメンバはそれぞれ次の内容を含む．

リスト1　メッセージループ

```
MSG msg;
   .
   .
   .
while (GetMessage(&msg, NULL, NULL, NULL))
    {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

リスト2　メッセージ構造体

```
/* Message structure */
typedef struct tagMSG
  {
    HWND        hwnd;
    WORD        message;
    WORD        wParam;
    LONG        lParam;
    DWORD       time;
    POINT       pt;
  } MSG;
```



画面　Windowsのメッセージ

C++ & OOP 用語集
構造体　必ずしも同一の型ではない複数のメンバが集合したデータ構造を指す．

・hwnd

メッセージのあて先．送り先のウィンドウのウィンドウハンドルがセットされる．ウィンドウハンドルとは，Windows中でそのウィンドウを一意に識別するための数値である．

・message

具体的なメッセージの内容．Windowsで発生するすべてのメッセージはWM_ではじまる名称がつけられている．windows.hファイルには，このメッセージ定義も含まれる．

・wParam，lParam

メッセージに関する付加情報．付加される情報はそれぞれのメッセージによって異なる．

・time

メッセージがメッセージキューに入れられた時間．

・pt

メッセージがメッセージキューに入れられたときのマウスの座標．

Windowsにおけるクラスとインスタンス

Windowsにおけるメインのオブジェクトであるウィンドウのデータ型を定義するものとしてウィンドウクラスがある．ウィンドウは，それぞれサイズ，画面上の表示領域，タイトルバーテキスト，カーソル，アイコン，バックグラウンドカラーなどの特徴をもっている．これらの情報を保持するのがウィンドウクラスである．

Windowsアプリケーションでは，CreateWindow関数が呼ばれたときに，ウィンドウクラスに基づいてウィンドウが描かれる．ウィンドウクラスは，WNDCLASSデータ構造体によって定義される．プログラム中でウィンドウクラスを登録すれば，Windowsで実行されているほかのすべてのプログラムがそのクラスを使えるようになる．同じアプリケーションをもうひとつ実行するときにも，同じウィンドウクラスが使えることになる．

ウィンドウクラスを登録するには，必要な情報をWNDCLASS構造体（リスト3）にセットして，RegisterClass()関数をコールする．WinMain()関数で最初に行うのは，ウィンドウクラスの登録である．if(!hPrevInstance)は，それが最初のインスタンスであるときだけという意味である．つまり，現在同じアプリケーションがすでに動作しているのなら，あとから動いたアプリケーションではもうウィンドウクラスを登録する必要はないということだ．

WNDCLASS構造体にセットする内容は次の通りである．

リスト3　WNDCLASS構造体

```
typedef struct tagWNDCLASS
  {
     WORD        style;
     LONG        (FAR PASCAL *lpfnWndProc)();
     int         cbClsExtra;
     int         cbWndExtra;
     HANDLE      hInstance;
     HICON       hIcon;
     HCURSOR     hCursor;
     HBRUSH      hbrBackground;
     LPSTR       lpszMenuName;
     LPSTR       lpszClassName;
  } WNDCLASS;
```

リスト4　ウィンドウクラスの登録

```
WNDCLASS  wc;

wc.style = NULL;
wc.lpfnWndProc = MainWndProc;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = GetStockObject(WHITE_BRUSH);
wc.lpszMenuName = "UserMenu";
wc.lpszClassName = "UserWClass";

return (RegisterClass(&wc));
```


C++ & OOP 用語集
再帰呼び出し　自分自身を呼び出す関数呼び出しのこと．

・**style**

ウィンドウクラスのスタイル．一般にCS_HREDRAW | CS_VREDRAWのように定義することが多い．CS_HREDRAWフラグはそのウィンドウの水平方向のサイズが変わったとき，CS_VREDRAWは垂直方向のサイズが変わったときに，Windowsクライアント領域全体を再描画する．

・**lpfnWndProc**

ウィンドウ手続き名を指定する．実際にメッセージを受け取って処理を行う関数名を指定する．

・**cbClsExtra**

クラス登録時に作成されるクラス構造体のコピーの最後に，追加する予備の空間サイズを指定する．通常は0をセットする．ここに値をセットして補助領域を確保し，情報をセットすると，このクラスのインスタンスはすべてその情報を共有できるため，データのカプセル化の手段としても利用できる．

・**cbWndExtra**

ウィンドウ作成時に作成されるウィンドウデータ構造体のコピーの最後に，追加する予備の空間サイズを指定する．通常は0をセットする．ここに値をセットして補助領域を確保して情報をセットすると，特定のインスタンスだけがその情報を保持できるため，データのカプセル化の手段として利用することもできる．

・**hInstance**

アプリケーションプログラムのインスタンスハンドル．Windowsはマルチタスク環境であるため，同じアプリケーションが2つ以上動作することもありうる．このとき，実行されているアプリケーションのひとつひとつを「インスタンス」といい，インスタンスを一意に識別するためにつけられる番号がインスタンスハンドルである．現在のインスタンスハンドルの値は，WinMain関数が実行されたときに，WinMain関数の引数hInstanceにセットされるので，その値をWNDCLASS構造体にセットしてやればよい．

・**hIcon**

このプログラムで使用するアイコンのハンドル．

・**hCursor**

このプログラムで使用するカーソルのハンドル．

・**hbrBackground**

ウィンドウのクライアント領域のバックグラウンドカラーを指定する．hbrというプレフィックスはブラシのハンドルを意味する．ブラシとはWindowsのグラフィックスの用語で領域を塗りつぶすための色つきのピクセルパターンのことである．Windowsには，標準で何種類かのブラシが用意されている．GetStockObject()関数は，ブラシのハンドルを返す関数である．ここで指定されているWHITE_BRUSHは，白いブラシ，すなわち背景を白で塗りつぶすという標準的な選択である．

・**lpszMenuName**

そのウィンドウクラスで使用するメニューの名称．メニューを使わないときは，NULLまたは0を指定する．

・**lpszClassName**

ウィンドウクラスの名前．プログラム中で使用するウィンドウクラスが1つなら，プログラム名を使ってもよい．

Windowsにおけるクラス階層

オブジェクト指向プログラミングには，クラスの抽象化の程度によってクラスを階層化し，下位のクラスが上位のクラスの性質を継承することによって，クラス間の情報を共有する考え方がある．Windowsにおけるメインのオブジェクトであるウィンドウの階層化は，正統的なオブジェクト指向型システムに見られるほど強力には実現されていないが，これに近いものとしてウィンドウのサブクラス化がある．ウィンドウのサブクラス化は，Windowsアプリケーションにおける親ウィンドウと子ウィンドウの関係とはまったく別のものである．

ウィンドウをサブクラス化すると，あるウィンドウに対するメッセージ処理関数を別のものと置き換えて，プログラム中のいくつかのメッセージを横取りして処理し，残りのメッセージはもとの関数にもどしてやることが可能になる．具体的には，MakeProcInstance()関数で，サブクラス化した新たな関数が正しいデータセグメントにアクセスできるようにし，次にGetWindowLong()関数でもとの関数のアドレスを保存して，SetWindowLong()関数をコールすればよい．サブクラス化された関数の最後でCallWindowProc()をコールすれば，処理をしなかった残りのメッセージはもとの関数にもどして処理させることができる．

サブクラス化は，特定のインスタン

C++ & OOP 用語集

最適化　高級言語において無駄なコードを省き，プログラムサイズを小さく，または実行速度を速くするためにプログラムを加工すること．

スの機能を制限する場合などに使われる．たとえば，画面上のコントロール間をTABキーで移動できるよう，標準のコントロールあてのメッセージを横取りするサブクラスを作成したりする場合に有効である．

オブジェクト指向プログラミング言語による開発

処理系は2種類に分類できる

Windowsとオブジェクト指向プログラミングをめぐるテーマの最後として，オブジェクト指向型プログラミング言語によるWindowsアプリケーションの開発について取りあげよう．

現在，マイクロソフト社が提供するアプリケーション開発環境は，MS-CまたはMASM＋Windows開発キット（SDK）の組み合わせである（日本では，Windows同様SDKは各パソコンメーカーが販売している）．Windowsアプリケーションの構造は，これまで説明してきたように，オブジェクト指向プログラミングの要素が多く取り入れられているが，実際のプログラム開発には，従来からある手続き型言語（C，アセンブラ）を使いなさい

COLUMN

新たなWindowsアプリケーション開発環境とオブジェクト指向プログラミング

マイクロソフト株式会社第1研究開発部言語課主務　**能祖裕樹**

MS-Windowsは，1.1，2.0，2.1，2.11とバージョンアップして，今回の3.0で3回目の大きな更新が行われました．今回のバージョンでもWindows用のアプリケーションの開発は難しいと思います．弊社では，EXCELという製品を出していますが，あのようなWindowsアプリケーションを書くとしたら，DOS環境の似たようなアプリケーションを考えるよりも難しいことがたくさん発生します．単純にAPIを使ってアプリケーションの構築をすればいいというわけではなく，Windowsから呼ばれるかもしれない関数を自分であらかじめ定義しておくとか，マウスのイベントが発生したら何を行うかなど，プログラムの構築の仕方や，考え方がいままでとかなり違います．

DOSアプリケーションでは，アプリケーションを構築するうえで考えなければならないことはOSやハードウェアとのインターフェースについては，それほど多くありません．アセンブラを使ってハードの割り込みが入ってきたときに，その後の処理をどうするのか，ある特定の条件になってハードウェアが信号を送ってきたらどう処理すべきなのか，ということはありますが，通常のアプリケーションではそれほど悩まずに構築できます．Windowsアプリケーションは，そういったことを考えなくてはいけなくなってきたところが難しいところの1つだと思います．

現状のWindowsのAPIというのは，まだ完全にオブジェクト指向に対応しているというわけではありません．Windows APIに準拠したオブジェクト指向のライブラリは弊社でもまだ出しておりません．実は，サードパーティ様からそういったライブラリがいくつか出ていますが，まだ一般的ではありません．OEMメーカーさんから供給されるSDKに含まれているライブラリも，現状ではまだ通常の手続き型の処理をするものです．手続き型では，ライブラリのほうから，逆に何かシグナルが返ってくる場合を想定しなければならないというのが，非常に難しいでしょう．ウィンドウひとつとってみても，どういうかたちにするかというのを引数で定義しています．インターフェース部分を構築するのは，ダイアログエディタなどを使えば簡単なのですが，ひとつひとつのルーチンワークを書くところは，まだまだ手続き型で処理をしなければなりません．

難しい点は，まだまだいろいろとありますが，将来の話をしますと，そのWindowsのAPI自体をクラスラ

C++ & OOP 用語集
参照型　関数に渡される変数そのものを操作するためにCでは複雑な手続きを必要としたが，C++は参照機能を利用し実際の変数を渡して変更するだけである．

というわけだ．これに対し，オブジェクト指向のプログラミング言語を使ったアプリケーション開発環境を提供しようという動きが徐々に現れつつある．

オブジェクト指向型プログラミング言語をWindowsアプリケーションの開発に利用するメリットは，オブジェクト指向プログラミングというパラダイムによって，開発環境が統一されて見通しがよくなることである．オブジェクト指向型のプログラミング言語は，オブジェクト，クラス，インスタンスなどの概念を言語仕様としてそなえているため，これらの概念を取り入れているWindowsのGUIを構築するには適しているといいかえてもいい．

現在，パソコン用として販売されているWindows 3.0対応のオブジェクト指向プログラミング言語処理系は2種類に分類できる．MS-Cと同じくMS-DOS環境でエディタを使って記述し，コンパイル，リンクして実行形を作成するものと，ソースのエディットからコンパイル，リンク，デバッグまですべてをWindows上で実行するものである．

前者には，SDKを必要とするものもある．SDKを使用することを前提とすれば，CやPascal系の処理系がWindows 3.0に対応するのはそれほど困難なことではない．MS-Cとコンパチブルなオブジェクトが出力できればいいのだから．後者には，WindowsのGUIを生かして，開発環境そのものもオブジェクト指向の考え方で統一しようと試みている処理系もある．

・C++

C++は，C言語にオブジェクト指

イブラリのかたちで供給することを考慮していきたいと思います．また，現在弊社の主力言語製品は，まだ手続き型の言語製品ばかりですけれども，Microsoft Cを筆頭に，OOPをサポートしたツールを製品化していきたいと考えています．

もうひとつのアプローチは，グラフィカルユーザーインターフェース上での開発環境を提供していくことです．現状では，DOSの世界でProgrammer's WorkBench，もしくはQuickシリーズのエンバイロメントを使ってアプリケーションを作成し，実際の実行ファイルはWindows上で実行を確認しなくてはなりません．先日，米国で出荷されたVisualBASICは，新しい開発環境のひとつの解答といえるでしょう．VisualBASICは，イベントドリブンなプログラミングを可能にする言語仕様をもっています．VisualBASICでは，メニューやダイアログボックス，各種ボタン，テキストボックスなどのオブジェクトを画面上にビジュアルに配置し，それらのオブジェクトに対して発生したイベントの処理ルーチンをプログラミングするのです．VisualBASICは，基本的なBASICの仕様のほかに，ウィンドウや他のウィンドウアプリケーションを管理する機能ももっています．独立したウィンドウアプリケーションの構築はもちろんのこと，他のウィンドウアプリケーションの処理機能を利用できるユーザーインターフェース構築ツールとしても活躍するでしょう．

その思想は，VisualBASICのみならずほかの言語にもあてはまります．違うアプローチですが，たとえばCASEツールみたいなものを搭載したり，単純にグラフィカルユーザーインターフェース上で，Quickシリーズのようなエンバイロメントが出てきて，そこでソースプログラムを書いて，実行すると別ウィンドウとして実行されて，デバッグもその環境でできるような……，そうしたツールの開発も今後は検討していきたいと思っております．

Windowsアプリケーションが開発できるまとまったパッケージとしては，一般的にBorland C++がいちばん最初のように思われますが，実際にはタッチの差で米国マイクロソフト社のMicrosoft C 6.0 With SDKのほうが先だったはずです．マイクロソフトも今後はSDKをバンドルしたパッケージというのを考慮していくつもりです．日本では，価格のギャップを埋められないという理由でWith SDKは製品化しません．私どもの1年の期末というのは6月末で，7月からちょうど92年度になりますが，92年度のうちにはグラフィカルユーザーインターフェースサポートのプログラミングツールを，製品として何種類か出したいと思っています．また，オブジェクト指向型のプログラミングツールというものも出荷したいと考えています．オブジェクト指向型プログラミングツールのほうは，みなさんおまちかねのものになるでしょう．

C++ & OOP用語集
持続期間　記憶クラスと密接な関係があり，宣言された識別子がメモリ内に割り当てられた実体のあるオブジェクトを保持している期間を指す．

向プログラミングの要素を取り入れて拡張した言語で，AT&Tのベル研究所でBjarne Stroustrupによって開発された．C++の特徴は，クラスの概念を取り入れていることである．ユーザーは，新たなデータ型としてクラスを宣言することができる．C++でいうクラスは，Cの構造体の概念を拡張したものだが，構造体と大きく違うのは，クラスのメンバとして関数をもつことができる点である．

パソコン用C++処理系でWindowsアプリケーションの開発ができる製品としては，Zortech C++，Borland C++があるが，いずれもまだ日本語版は発売されていない．Zortech C++でWindowsアプリケーションを開発するには，SDKを必要とする．Borland C++には，SDKに相当するライブラリや，Whitewater Resource Toolkitも含まれる．どちらの製品もソースエディットから実行形の作成まではMS-DOS上で行う．

・Smalltalk

Smalltalkは，Xeroxパロアルト研究所(PARC)のLearning Research Groupeによって，1970年代前半から研究されてきた成果として実現された．Smalltalkは，プログラミング言語というより，オペレーティングシステム，言語処理系，アプリケーションプログラムのすべてをオブジェクト指向というかたちでモデル化したものだといえる．PARCの研究は，Windowsをはじめとする今日のGUIで採用されているアイデアを生みだした．つまり，ビットマップディスプレイとマウスを使った対話型のユーザーインターフェースをもったワークステーション，Altoを実現させたのである．Smalltalk上でのプログラミングは，ブラウザで新しいクラスを作成し，メソッドをつけ加えることによって行われる．

本来，ワークステーション用であったSmalltalkシステムをパソコン用として実現したのがdigitalk社のSmalltalk/Vであり，そのWindows対応版としてSmalltalk/V for Windowsという製品がある．Smalltalk/V for Windowsは，それ自体Windowsアプリケーションである．開発のすべてのフェーズをWindows上で行うことが可能で，SDKも不要である．Class Hierarchy Brouserというウィンドウ中で，Smalltalk/Vがあらかじめもっているメソッドを変更したり，新たなメソッドを付加することによってアプリケーションを開発できる．

・Actor

Actorは，Whitewater社が販売するWindowsアプリケーション開発システムである．「Actor」という製品名自体，オブジェクト指向プログラミングの本でよくとりあげられる「Actor理論」からとったものであろう．

Actorの環境は，Smalltalkとよく似ており，Actorそのものが Windowsアプリケーションであるため，開発のすべてのフェーズをWindows上で実行できる．プログラマは，クラスライブラリをブラウジングし，新しいクラスを作成して，メソッドを記述する．Actorの特徴は，あらかじめWindowsアプリケーション開発用として，多数のクラスをそなえていることである．WindowsのGUI作成用としては，Window，Dialog，Controlなどのクラスが用意されている．Controlに関するクラス階層は，Actorではリスト5のように定義されている．たとえば，スクロールバーを作成するには，Scrollbarクラスにnewというメッセージを送ってインスタンスを生成する．そして，このインスタンスにshowメッセージを送れば画面に表示されるという具合である．

リスト5　Actorにおけるcontrolに関するクラス階層

```
Control
  Button
  Edit
  ListBox
    ClassList
    MethodList
    VarList
  Scrollbar
```

現在，Windowsのグラフィカルユーザーインターフェースは注目を集めており，そのアプリケーション開発の需要も高まっていることを考え合わせると，Windowsアプリケーションの開発にはこれからますますオブジェクト指向言語による開発環境が提供されていく傾向が強まると予想できる．

（編集部）

C++ & OOP 用語集
スコープ　適用範囲のこと．識別子のスコープは，その識別子をそのオブジェクトにアクセスするために使用できるプログラムの一部分である．

特集
はじめての
オブジェクト指向
プログラミング



Macintosh環境における オブジェクト指向

クラスライブラリとの格闘はまぬがれない

新居雅行

Macintosh のアプリケーション作成は難しいと長い間いわれ続けてきたが，これは基本的にイベント駆動型である Macintosh のプログラムと従来の手続き型的な考え方で作成するプログラムの間にギャップがあったためである．また，Macintosh のもつ GUI(Graphical User Interface) もプログラムを作る上での大きな障害になっていた．
しかし，現在では MacApp などに代表される高機能なクラスライブラリが提供されるようになり，またオブジェクト指向パラダイムを用いたプログラミングに対する理解の広まりと相まって，Macintosh のプログラミングは容易であるとの認識か広まりはじめている．
しかしながら現実には，オブジェクト指向プログラミングを身に付け，実施するには，いかに Macintosh が“人にやさしい”パソコンであったとしても，かなりの時間と労力が必要となる．
オブジェクト指向パラダイムを理解するために，まず，Macintosh の GUI をオブジェクト指向をとおして眺め．次に，オブジェクト指向プログラミングをとても身近にしてくれた HyperCard によるプログラミングを考え，さらに Macintosh で利用できる代表的なクラスライブラリである TCL（THINK Class Library）について，その利用方法を学ぶうえでの問題点を述べる．

オブジェクト指向の代表選手といえば，まずSmalltalkマシンが引き合いにだされるが，最近ではMacintosh（以下，マック）がかつぎだされることも多くなったようだ.

また，MS-Windows Ver.3.0も登場し，世の中の流れがGUI（Graphical User Interface）に向かっているというのもどうやら当たっていそうである．このWindows, Samlltalk, マックのもつGUI環境とオブジェクト指向との間には，深い関係がある．実際，オブジェクト指向における“クラス”の説明の材料として，四角形やポリゴンなどのグラフィックスオブジェクトが使用されることも少なくない．「形」をもったオブジェクトというのは，誰でも容易に理解できる実例なのである.

しかし，だからといって“GUI＝オブジェクト指向”であるとは限らない．マックに限らず，どんなシステムでもいろいろな表情をもっている．オブジェクト指向的である場合もあれば，かぎりなく手続き型に近い場合もある．つまり,「オブジェクト指向であるかどうか」との問いに，単純にイエスかノーかと答えられないというのが一般的なのだ．マックは，かなりオブジェクト指向的なシステムであるが，やはりオブジェクト指向に合致しない部分もある.

ここでは，マックのもつユーザーインターフェースをオブジェクト指向的に解説するとともに，ハイパーカードやTHINK系の言語処理ソフトウェア，さらにクラスライブラリについて紹介し，マックとオブジェクト指向との関わり合いについて見てみたい.

ユーザーインターフェースとオブジェクト指向

ファインダ（Finder）は，マックのシステムに付属しているファイル管理プログラムで，基本的にシステム立ち上げ時に起動するソフトウェアである.

ファインダでは，図1にあるようにデスクトップの上にウインドウがあり，ウインドウの中にはアイコンがある．アイコンは，ボリューム内のファイルの存在を示すものである．このアイコンをオブジェクトとしてとらえて，オブジェクト指向プログラミング風に記述してみよう.

アイコンを画面上で扱うには，どのような処理が必要だろうか？　まずアイコンは，アイコンの位置を変更したり，クリックしたら選択されるなど，さまざまな状況に対応できなければならない．そこで図2のように，アイコンのオブジェクトを考えてみた．オブジェクトは，次のような形式で記述している.

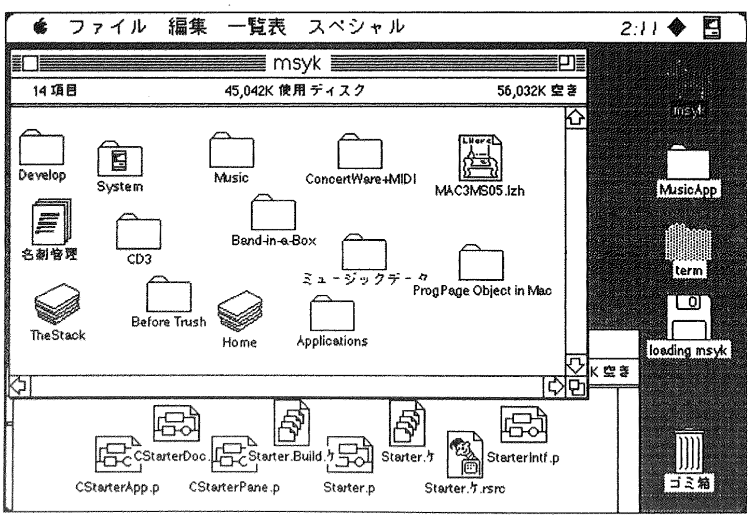
クラス名（オブジェクト名）
インスタンス変数
……
メソッド
……：……

ここで，インスタンス変数はそのオブジェクトのもつデータ，メソッドはそのオブジェクトのもつ機能と思えばよい．もちろん，これは厳密な定義ではなく，これをもとにプログラミングができるというレベルではない.

話を簡単にするために，メッセージのセンダー(送り手)はユーザーということにしよう．ユーザーは，マウスを使ってメッセージをアイコンに対して送ることができるというわけだ．たとえば，アイコンを選択するという行為は，実際には「マウスでクリックする」ということになるのだが，上記のようなオブジェクトモデルでは,「アイコンに対してメッセージ“Select”を送る」ということになる.

マウスでアイコンをクリックするという簡単なことをわざと複雑に説明しているように思われるかもしれないが,

図1　オブジェクト指向的な操作でファイル管理を行うFinder



C++ & OOP用語集
スコープアクセス演算子　::（コロン2つ）によって，グローバル名へのアクセスが，ローカルな再宣言によって隠されている場合でも可能になる.

オブジェクト指向というのは一種の“ものの見方”そのものなので，どうしても話が抽象的になってしまうのである．

さて，ファインダでは，アプリケーションの文書ファイルをダブルクリックすることで，アプリケーションが起動し，その文書ファイルをオープンすることができる．この「文書のアイコン」という新しいオブジェクトを先ほどと同じように記述してみると，図3のaのようになる.「文書のアイコン」も，やはりアイコンの図柄，アイコンの位置などのデータをもち，移動，削除などが可能である．これらは,「アイコン」というオブジェクトと共通したものだ．さらに,「文書のアイコン」には図3のaの中で文頭に●をつけたようなインスタンスやメソッドが追加されている．

図2　アイコンをオブジェクトとして記述する

```
アイコン
    インスタンス変数
        アイコンの図柄
        アイコンの位置
        ファイル名
        作成日
        ファイルサイズ
        ロックされているかどうか
        選択されているかどうか
    メソッド
        Select     ：アイコンを選択する
        Move       ：アイコンを移動する
        Erase      ：アイコンを削除する
        Deselect   ：アイコンの選択を解除する
        GetInfo    ：ファイルの情報を取り出す
```

ここで，あるオブジェクトの性質を引き継いで新たなオブジェクトを作るという，いわゆるオブジェクト指向の“継承（inheritance)”を導入して文書のアイコンを定義すると図3のbのようになる．“←アイコン”のように“←”を用いてそのオブジェクトの上位クラスを示している．ユーザーがマウスでダブルクリックするという操作は,「文書のアイコンにメッセージ“Boot”を送る」ということになるわけである．

図3　文書ファイルのアイコンをオブジェクトとして記述する

```
a.単純にオブジェクトを作る

文書のアイコン
    インスタンス変数
        アイコンの図柄
        アイコンの位置
        ファイル名
        作成日
        ファイルサイズ
        ロックされているかどうか
        選択されているかどうか
       ●属するアプリケーション
    メソッド
        Select     ：アイコンを選択する
        Move       ：アイコンを移動する
        Erase      ：アイコンを削除する
        Deselect   ：アイコンの選択を解除する
        GetInfo    ：ファイルの情報を取り出す
       ●Boot       ：アプリケーションを開く

b.アイコンのサブクラスとして定義してみた

文書のアイコン←アイコン
    インスタンス変数
        属するアプリケーション
    メソッド
        Boot       ：アプリケーションを開く
```

次にフォルダはというと，たとえば図4のように記述できる．ほんとうはもっといろいろメソッドを用意しないといけないのだろうが，とりあえず「アイコン」とほぼ同じ性質をもつサブクラスということにしよう．

フォルダのアイコンに対しても，文書ファイルのアイコンに対しても，GetInfoによって情報が得られることになっている．いずれのアイコンに対しても，アイコン（つまりファイルかフォルダ）の作成日やロックがかかっているかどうかといった共通の属性があり，これらがGetInfoでレポートされる必要がある．

フォルダのアイコンに対してユーザ

C++ & OOP 用語集
静的メンバ　記憶クラス指定子staticを，データおよび関数メンバのクラス宣言に使用することがある．このメンバを静的メンバという．

ーが知りたい情報となると，フォルダ内のファイルの数やすべてのファイルサイズの合計などだろう．文書のアイコンだと，そのファイルの大きさやもとのアプリケーションの名前などが知りたい．ここで重要なのは，フォルダのアイコンであろうと，文書のアイコンであろうと，情報を得るにはどちらも同じく“GetInfo”というメッセージをアイコンに対して送ってやるという点である．操作はまったく同じなのである．現にファインダでは，フォルダであろうが文書のアイコンであろうが，「ファイル」メニューの「インフォメーション」を選択することで情報をウインドウに表示する．つまり，実際に実行される処理手続きは異なっていても，処理を実行させるために送るメッセージは共通なのである．

これらのメソッドの処理手順は，図5のようになるだろう．

ここで見てきたように，マックユーザーがふだん何気なく操作しているアイコンも，その機能と動作はオブジェクト指向という考え方にピッタリとマッチしている．アイコンに限らずウィンドウやプルダウンメニューなど，GUIに不可欠な要素をプログラミングしようとしたとき，オブジェクト指向という考え方はごく自然なものになるのである．

ハイパーカードに見るオブジェクト指向

ハイパーカード（HyperCard）は，オブジェクト指向を強調してはいないものの，至るところでオブジェクト指向の特徴的な性質が見られる．ハイパーカードそのものについての詳しい説明は行わないが，ある程度基本的なところから説明していこう．

まず，ハイパーカードの文書ファイルはスタックと呼ばれるもので，ハイパーカードという名前どおり，カードがいくつか組み合わさってスタックが形成される．最新のバージョンのハイパーカードではカードの大きさをある程度任意に設定できるが，Ver.1ではカードの大きさは一定だった．

カードはバックグラウンドとフォアグラウンドが重なって見えており，いくつかのカードでバックグラウンドは共通に使用できる．このいずれのグラウンドにも，ボタン，テキストフィールド，ビットマップの絵を配置できる．たとえば，バックグラウンドに「住所」や「氏名」を入力するテキストフィールドを設定すれば，名詞管理のカード型データベースのようなものが作れるわけだ．

ハイパーカードのボタンやテキストフィールドは，オブジェクトとして考えられる．もちろん，フォアグラウンドもバックグラウンドも，そしてカードそのものやスタックも，やはりオブジェクトである．ビットマップの絵も，どちらかのグラウンドにあるオブジェクトと考えることが可能だ（図6）．

図7のようなスタックを通常の言語で作るとなると，ソースの中に，case文がいっぱい出てきて，あるスイッチを押したときにはどうこう，別のスイッチの場合は...などというリストにするが，ハイパーカードはちょっと（えらく？）違う．というのは，ボタンを押したときの処理は，ボタンそのものにスクリプトとして記述するからである．このスクリプトを記述するための

図4　フォルダをアイコンのサブクラスとして定義

```
フォルダのアイコン←アイコン
    インスタンス変数
        フォルダ内のファイル数
    メソッド
        （追加なし）
```

図5　サブクラスのメソッドの処理手順

```
アイコンのGetInfo
 1.作成日を取り出す
 2.属性を取りだす
 3.表示する

文書のアイコンのGetInfo
 1.アプリケーション名を取り出す
 2.アイコンのGetInfoを呼び出す

フォルダのアイコンのGetInfo
 1.フォルダ内のファイル数を求める
 2.フォルダ内のファイルのサイズの合計を求める
 3.アイコンのGetInfoを呼び出す
```


C++ & OOP 用語集
多重継承　2つ以上の上位クラスを継承すること，またその仕組

言語が，ハイパートーク（HyperTalk）と呼ばれるものである．

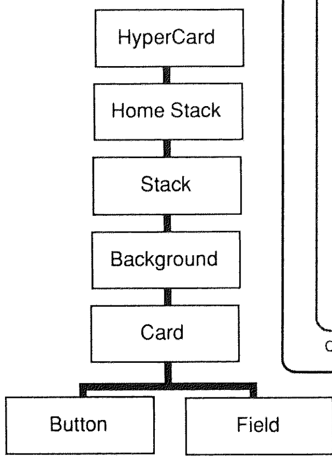
ハイパートークは，図7にあるように，case文的ではなくイベント駆動的な記述をする言語だ．このボタンに設定するスクリプトは，オブジェクト指向プログラミング言語でのクラス定義中のメソッドの記述によく似ている．

このようなボタンやテキストフィールドなど，オブジェクトそのものにスクリプトを書くことがすでにオブジェクト指向であり，オブジェクト隠ぺいという性質をもっているといえる．メッセージセンディングについては，たとえばユーザーがマウスボタンを押したとき，マウスカーソルが指しているオブジェクトに対して“mouseDown”というメッセージが送られ，マウスボタンをはなしたときには“mouseUp”というメッセージが送られる．このように，ユーザーからのアクションに対して，オブジェクトへのメッセージがさまざまな形で送られるわけだ．

ボタンは，メニューにある「新規ボタン」というアイテムを呼び出すことで作成できる．オブジェクトの作成に際して，テキストでクラス定義的なことをやる必要はないのである．メソッドはテキスト（スクリプト）で記述するのだが，ボタンの位置などのオブジェクトの属性については，画面上でマウスを用いて設定できる．

さらに，オブジェクトへのメッセージは継承される．継承の仕方は，まず最下位のボタンへあるメッセージが送られた場合，ボタンにそのメッセージを処理するメソッドがあるかどうかを調べる．もしあればその処理を行うが，なければ1つ上位のカードにメソッドがあるかどうかを調べる．こうして，最終的に最上位のハイパーカードにまでメッセージが継承されるのである．

図6　ハイパーカードの概念とクラス構造



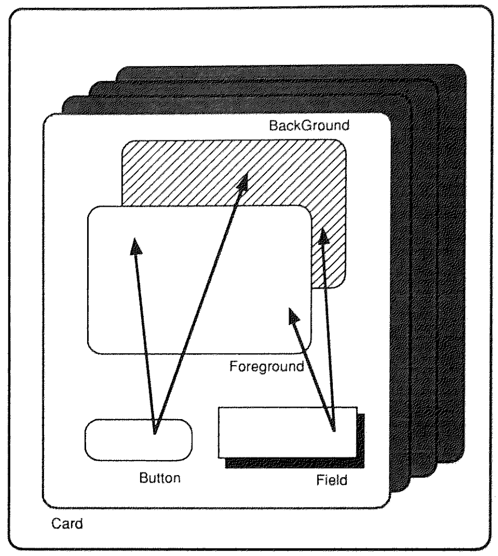
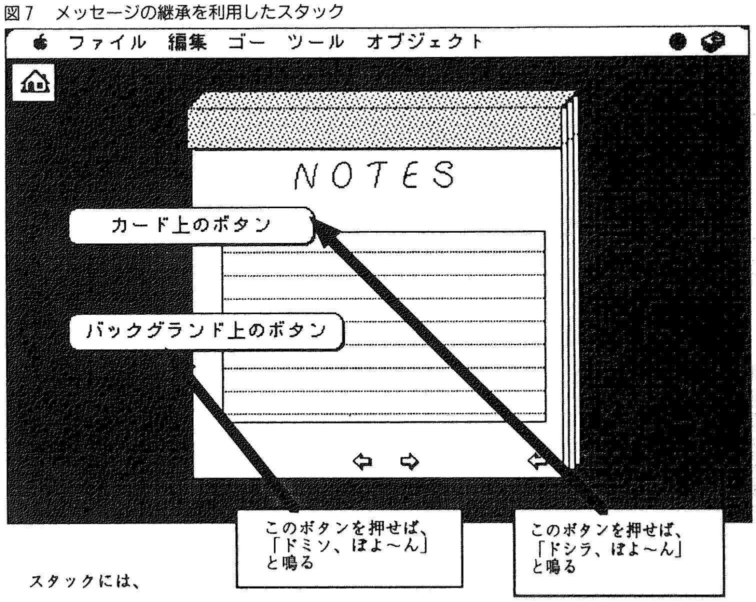


図7　メッセージの継承を利用したスタック



スタックには、

```
on mouseUp
 play "boing"
end mouseUp
```

というスクリプトが含まれている。

「カード上のボタン」には、

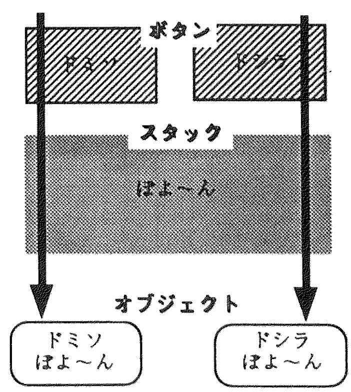
```
on mouseUp
 play "harpsichord" temp 150 "c4q e g"
 pass mouseUp
end mouseUp
```

というスクリプトが含まれている。

「バックグランド上のボタン」には、

```
on mouseUp
 play "harpsichord" temp 150 "c4q b3 a3"
 pass mouseUp
end mouseUp
```

というスクリプトが含まれている。



多重定義　Cでは同名の関数を定義することができなかったのに対し，C++では同名でも扱うデータ型が異なる関数を定義することができる．これを多重定義という．

ボタンにあるスクリプトを処理し，さらに上位クラスにある同じメッセージに対応するスクリプトにメッセージを渡すようにするには，passというコマンドを使って記述する必要がある．

しかしながら，正直な話，このハイパートークの書き方には，はじめはかなりとまどった．すべてをオブジェクト指向的に行おうとすると，どうもあるところで破綻をきたしてしまうのである．そこで，いろいろなスタックを解析してみたところ，あまりオブジェクト指向的でないものも見受けられた．結局，手続き型的に考えられる部分は普通（？！）に作ればいいのである．

クラスライブラリの利用と問題点

マック上で汎用言語を使ってオブジェクト指向プログラミングをやるとなると，MacAppやTCL（THINK Class Library）などが必要になる．

MacAppは，MPW（Macintosh Programmer's Workshop）上で動くObject PascalやC++のライブラリである．現在は，E.T.O.というCD-ROMに必要なツールが全部詰まったセットでも販売されている．一方，TCLは，THINK　CやTHINK Pascalに付属しているライブラリである．また，TCLはライブラリだが，オブジェクト（これはマシン語という意味での）ファイルではなく，CまたはPascalで記述されたソースファイルである．

MacApp，TCLのいずれを使用しても，マックのアプリケーションが作成できることはいうまでもない（TCLとMacAppの比較は，インターフェース誌1991年4月号に詳しい記事がある）．

オブジェクト指向の入門書を一生懸命読んで，さてプログラミングをと思っても，はたと手がとまってしまったという経験がないだろうか？　オブジェクト指向の基本的なことがわかっているだけでは，プログラムを作ることはできないのである．

オブジェクト指向の概念を説明する例に，グラフィックスがよく使われることは前にも書いたが，ほかにもメニューだとかあるいはウインドウなどのビジュアル階層を例にしてオブジェクト指向について解説することが多い．

このような学習をしたあとで，じゃあプログラミングだといって，例にあげられていたようなビジュアル階層やらを作っていくことはまずありえない．なぜなら，現実には，学習時に例題として出されたようなものは，すでにライブラリに入っているからである．

そして，このような既存のライブラリを使う場合には，オブジェクト指向うんぬんよりも，そのライブラリをどのように使うかという，ライブラリの使い方を学習しなければならない．つまりプログラマは，いきなり，高度に階層化された複雑きわまりないクラスライブラリを相手に格闘を演じなければならないのである．

もちろん，MacAppやTCLなどのクラスライブラリは，十二分に練られた結果作られたものなのだが，あのような深い階層，バーテンダーやクラスターなどの見たこともない（仮想クラスだからあたりまえ？）オブジェクトなどが登場し，はじめのうちはまるでダンジョンをさまよっているような気分になる．あたかも，ベートーベンやモーツァルトなどの古典に親しんだ耳に，いきなりシュトックハウゼンやジョンケージ！　という感じである．

オブジェクト指向のうえに組み上げられたライブラリを理解し，消化するには，やはりそれなりの時間と努力がいるようだ．

たとえば，TCLでのオブジェクト指向プログラミングは，いきなり1からクラスを定義するのではなく，クラスの階層構造の決まったところにサブクラスを定義するところからはじまる．つまり，Starterというフォルダに入っている一連のファイルを複写して，別のファイル名に変えるのである．このStarterは，文書ウインドウを表示する以外ほとんど何もしないのだが，クラスライブラリのほとんどのファイルが読み込まれている．

はじめてStarterを見ると，そのすごいクラス階層に当惑してしまうかもしれない．なにせ，このクラス構造の中にうまく整合性がとれるようにしてアプリケーションを組み立てなければならないのだから……．とにかく，まずソースファイル名を書きなおすというところからはじめ，規定されたクラスに仕様を満たすようなメソッドを記述していくことである．ここから，TCLを利用したオブジェクト指向プログラミングがはじまるのだから．

オブジェクト指向での開発にはブラウザが必須

また，オブジェクト指向プログラミ

C++ & OOP用語集
多態性　同じ名前の関数を呼び出すことで，オブジェクトごとに異なる機能を実現すること．

ングの環境でよくいわれるのが，優秀なクラスブラウザがあるとないとでは，プログラミングの効率がまったく違ってくるということだ．この点でいうと，THINK Cの現在のバージョンにはブラウザがないし，デバッガもこれまでの変数のモニタ程度の機能しかサポートしていない．次のバージョンで，これらの機能が追加されることを期待するしかない．

THINK Pascalにはクラスブラウザが付属し，さらに変数のモニタでSELFという変数が自動的に現われるなど，そこそこの機能が備えられている．しかし，開くことができるソースファイルのウインドウ数に制限があり，大量のソースでビルドされているTCLのプログラムをトレースしていると，すぐにウインドウを開くことができないという警告にあい，ウインドウを閉じなければいけなくなるのが難といえる（図8，9）．

マックはオブジェクト環境だとはいえ……

結局，マックを使っていて，そのあとにオブジェクト指向を勉強すると，ともすれば「オブジェクト指向ってマックのソフトそのものじゃないか」的な見方になってしまいがちである．これが教育効果なのか，洗脳なのかはともかく，マックがオブジェクト指向に向いたパソコン環境であることだけは確かなようだ．

しかし，オブジェクト指向について書かれた本にあるように，オブジェクト指向でプログラミングをしたとたんに，いとも簡単にマックのアプリケーションが書けるようになるなどというのは大まちがいである．クラスライブラリを相手にするのは，Inside Macintoshと格闘したのと同様な労力が必要になるということだけは覚えておいてもらいたい．

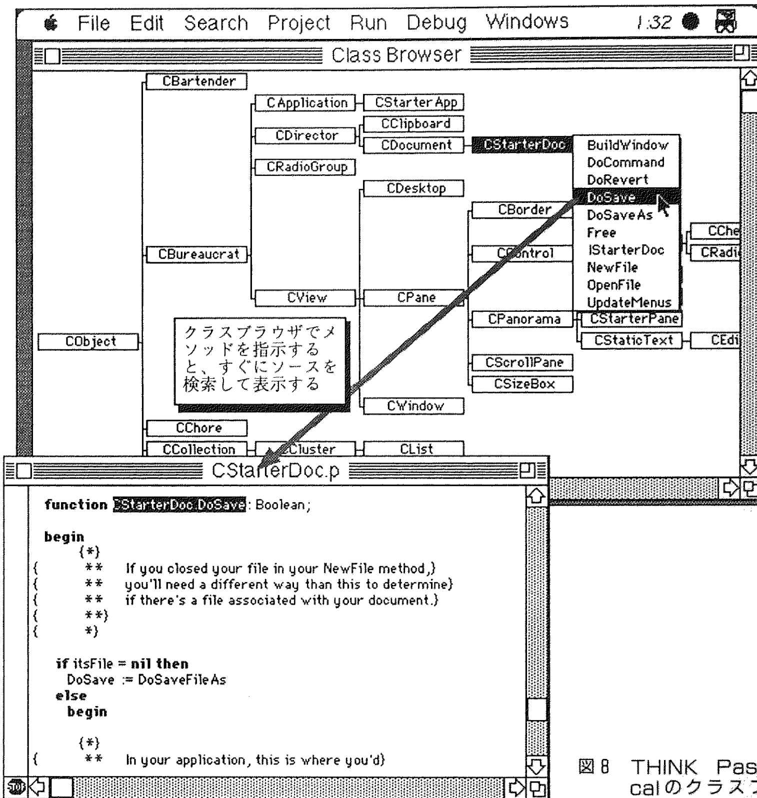


図8　THINK Pascalのクラスブラウザ

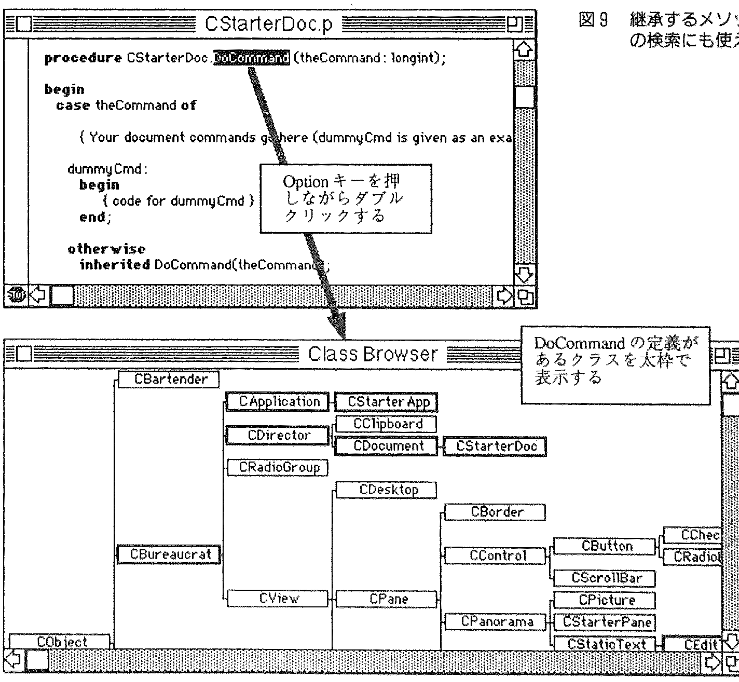


図9　継承するメソッドの検索にも使える

C++ & OOP 用語集

抽象クラス　アブストラクトクラスともいい，最低ひとつの純粋仮想関数をもつクラスのこと．仮想関数は純粋指定子を用いることにより「純粋」と定義される．



特集
はじめての
オブジェクト指向
プログラミング

Turbo Pascal 6.0に見る オブジェクト指向

オブジェクト指向ライブラリTurbo Visionの実体

編集部

米国 Borland 社が発売している Turbo Pascal は，MS-DOS 上で動作する安価で高機能な Pascal コンパイラとして世界中に多くのファンをもつ人気商品である．この Turbo Pascal がバージョン5.5になった際に，オブジェクト指向プログラミングの機能を取り入れてファンを驚かせたのは記憶に新しい．しかも，この機能拡張は，たった 4 つの予約語を付加することで実現されたのである．
現在発売されているバージョン6.0では，このオブジェクト指向プログラミングの機能が強化され，さらに Turbo Pascal が備えている IDE（統合開発環境）のような，オーバーラッピングウインドウを備えたアプリケーションを作成するためのクラスライブラリである Turbo Vision が追加された．実際，Turbo Pascal の IDE は，Turbo Vision を用いて開発されたということだ．このことは，Turbo Vision かたんなるオモチャではなく，実際のプログラミングに使える実用的なツールだということを示している．
Turbo Vision を備えた Turbo Pascal 6.0は，現在 PC-9801上で動作するもっとも安価で強力なオブジェクト指向プログラミング言語処理系といえる．この Turbo Pascal 6.0によるオブジェクト指向プログラミングと，Turbo Vision について紹介する．

オブジェクト指向はどのように実現されているのか

まず最初に，Turbo Pascal 6.0において オブジェクト指向プログラミングがどのように実現されているかという点について，トピックをあげながら紹介してみたい．オブジェクト指向プログラミングそのものについての説明は，本特集の他の章を参考にしてもらいたい．

オブジェクト

Turbo Pascalにおけるオブジェクトは，レコード型とよく似たオブジェクト型として予約語objectを用いて定義される(リスト1)．また，objectに続く ()の中に上位のオブジェクト名を記述することで，そのオブジェクトを継承することができる．図1にオブジェクト型の構文図を示す．

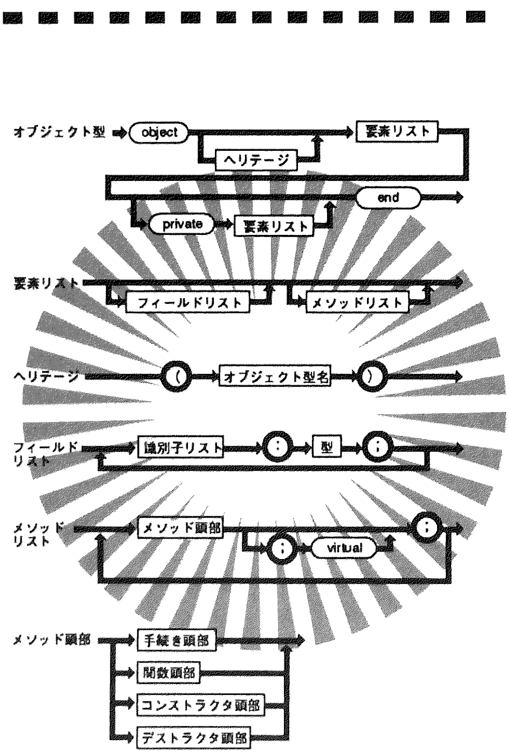
オブジェクト型は，ある型のデータを含むフィールドと，オブジェクトの操作を行うメソッドを含んでいる．

フィールドは，レコード型のフィールド同様with文および“.”を用いてアクセスすることができる．

メソッド

メソッドは，オブジェクト内で宣言される関数または手続きである．メソッドは，まずオブジェクト型の定義の要素リストの中で，procedureまたはfunctionとして宣言され，次に，オブジェクトの定義の外側で実際に，procedureまたはfunctionとして定義される．また，メソッドがオブジェクトの外側で定義されるときには，メソッド名の前にそのメソッドを含むオブジェクトの名前と“.”を記述する（リスト2）．

図1　オブジェクト型の構文図



リスト1　オブジェクトの定義

```
type
    test_1 = object          { オブジェクトの定義 }
        A, B : Integer;
        procedure Init(NewA, NewB : Integer);
    end;

test_2 = object(test_1)      { オブジェクトの継承 }
    C : Boolean;
    procedure Init(NewA, NewB : Integer);
    procedure Move;
end;
```

リスト2　メソッドの定義

```
type
    test_1 = object
        A, B : Integer;
        procedure Init(NewA, NewB : Integer);   { メソッドの宣言 }
    end;

procedure test_1.Init(NewA, NewB : Integer);    { メソッドの定義 }
begin
    A := NewA;
    B := NewB;
end;
```

C++ & OOP 用語集
動的オブジェクト　プログラム実行時に生成され，記憶域の割り当てがシステムの自由記憶領域から行われるオブジェクト．

ここで，オブジェクト型の要素の識別子のスコープは，そのオブジェクト型およびすべての下位型とすべてのメソッドにおよんでいる．したがって，要素の識別子はこれらの中でユニークなものでなければならない．

プライベート節

先ほどオブジェクト型のデータフィールドは，レコード型同様with文および“.”を用いてアクセスできると述べたが，これはオブジェクト指向に合致したことではない．なぜなら，オブジェクト指向では，オブジェクト内部のデータにアクセスできるのは，そのオブジェクトのメソッドのみでなければいけないからだ．オブジェクトの外から行えるのは，そのオブジェクトにメッセージを送ってメソッドを呼び出すだけ，というのが本来のオブジェクト指向プログラミングであり，こうすることで情報隠ぺい，カプセル化を実現しているのである．

そこでTurbo Pascalでは，オブジェクトの要素リストに予約語privateを用いて宣言することで，オブジェクトが宣言されたユニットの外からのアクセスを禁止することができるようになっている（リスト3）．プライベート宣言をしていない部分は，自動的にパブリックとなってオブジェクト外からのアクセスが可能になる．

なおプライベートとして宣言された要素の識別子のスコープは，そのオブジェクトが宣言されたプログラムまたはユニット内のみとなる．

バーチャルメソッド

Turbo Pascalのメソッドには，静的メソッドとバーチャルメソッドの2つがある．単純にメソッドを宣言しただけのものは静的メソッドになり，メソッド宣言の後ろにvirtual司令を付加すればバーチャルメソッドになる（リスト4）．

静的メソッドとバーチャルメソッドの違いは，そのメソッドの呼び出しがいつ解決されるかという点にある．静的メソッドでは，この解決がソースのコンパイル時に行われ，バーチャルメソッドではプログラムの実行時に行われる．後者のようにプログラムの実行時に呼び出しの解決を計る方法を，ダイナミックバインディング（dynamic binding）またはレイトバインディング（late binding）と呼んでいる．

バーチャルメソッドによって，オブジェクトの多態性を実現することが可能になる．多態性とは，オブジェクトの階層構造の中で，1つの名前で各オブジェクトがそれぞれのオブジェクトに適した異なるメソッドを呼び出すようになることをいう．

静的メソッドで，上位型のオブジェクトから継承したメソッドの内容を変更するには，下位型のメソッド宣言で上位型と同じメソッド名を使ってオーバーライド（再定義）する．このとき，メソッドヘッダを自由に書き換えて，引数と結果の型の異なるメソッドを定義することもできる．ただし，静的メソッドでは多態性は実現されない．

バーチャルメソッドをオーバーライ

表1
オブジェクトリファレンス（アルファベット順）

TApplication
上位オブジェクト：TProgram
フィールド：なし
静的メソッド：Init
バーチャルメソッド：Done
説明：TProgramを単に継承しただけのもの．違いはコンストラクタとデストラクタのみ．

TBackground
上位オブジェクト：Tview
フィールド：Pattern
静的メソッド：Inti, Load, Store
バーチャルメソッド：Draw, GetPalette
説明：単一パターンの矩形領域をもつ単純なビュー．

TBufStream
上位オブジェクト：TDosStream
フィールド：Buffer, BufSize, BufPtr, BufEnd
静的メソッド：Init
バーチャルメソッド：Done, Flush, GetPos, GetSize, Read, Seek, Truncate, Write
説明：TDosStreamにバッファリング機能を組み込んだもの．

TButton
上位オブジェクト：TView
フィールド：Title, command, Flags, AmDefault
静的メソッド：Init, Load, MakeDefault, Store
バーチャルメソッド：Done, Draw, GetPalette, HandleEvent, SetState
説明：名前と影をもったボックスで，押されるとコマンドを生成する．

TcheckBoxes
上位オブジェクト：TCluster
フィールド：なし
静的メソッド：なし
バーチャルメソッド：Draw, Mark, Press
説明：1～16個からなるチェックボックス．

TCluster
上位オブジェクト：TView
フィールド：Value, Sel, Strings
静的メソッド：Init, Load, DrawBox, Store

C++ & OOP 用語集
派生クラス　派生クラスはカンマで区切られた基本型リスト内に基本クラスをリストし宣言する．基本クラスのメンバはすべて継承される．

リスト３　プライベート節

```
type
  test_1 = object(上位型のオブジェクト)
      〈フィールド〉    { パブリックなフィールド }
      〈メソッド〉      { パブリックなメソッド }
  private
      〈フィールド〉    { プライベートなフィールド }
      〈メソッド〉      { プライベートなメソッド }
  end;
```

リスト４　静的メソッド／バーチャルメソッド

```
type
  test_1 = object
     A, B : Integer;
     procedure Init(NewA, NewB : Integer);    { 静的メソッドの宣言 }
  end;
  test_2 = object(test_1)
     C : Boolean;
     constructor Init(NewA, NewB : Integer);
     procedure do_1; virtual;                  { バーチャルメソッドの宣言 }
  end;
```

ドする場合は，必ずメソッドヘッダの内容を元のメソッドヘッダと一致させなければならない．また，上位型のオブジェクトでバーチャル宣言したメソッドと同じ名前の下位型のメソッドは，必ずバーチャルとして宣言しなければならない．つまり，静的なメソッドでバーチャルメソッドをオーバーライドすることはできないのである．したがって，いったんバーチャルとして宣言したメソッドの引数や型を変更することはできない．

オブジェクトの再利用性や拡張性を考慮するならば，メソッドは，常にバーチャルにすべきである．ただし，実行スピードやメモリ効率の点では静的メソッドのほうが優れているので，このような要求があるときには静的メソッドを使用するといいだろう．

コンストラクタ

バーチャルメソッドをもつオブジェクト型は，必ずコンストラクタをもたなければならない．また，個々のオブジェクトのインスタンスは，それぞれコンストラクタを呼び出して初期化しなければならない（リスト５）．

呼び出されたコンストラクタは，コンストラクタを呼んだインスタンスとオブジェクト型のバーチャルメソッドテーブル（VMT）との間の結合を作り出す．バーチャルメソッドテーブルとは，すべてのオブジェクト型がデータセグメントにもっている，オブジェクト型のサイズ，オブジェクト型のもつバーチャルメソッドを実現するコードへのポインタを含むテーブルである．つまり，オブジェクトのインスタンスからバーチャルメソッドを呼び出す際には，このバーチャルメソッドテーブルを介して間接的に実行コードが呼ばれることになる．これに対して静的メソッドは，オブジェクトのインスタンスから直接実行コードが呼び出される

バーチャルメソッド：Done, DataSize, GetData, GetHelpCtx, GetPalette, HandleEvent, Mark, MoveTo, Press, SetData, SetState
説明：同一の方法で反応するコントロールのグループ，抽象オブジェクト．

TCollection
上位オブジェクト：TObject
フィールド：Items, Count, Limit, Delta
静的メソッド：Init, Load, At, AtDelete, AtInsert, AtPut, Delete, DeleteAll, FistThat, ForEach, Free, FreeAll, LastThat, Pack, Store
バーチャルメソッド：Done, Error, FreeItem, GetItem, IndexOf, Insert, PutItem, SetLimit
説明：他のオブジェクトを含むすべての要素のコレクションを作成するための抽象オブジェクト．従来の配列，集合，リストよりも一般的な概念．

TDeskTop
上位オブジェクト：TGroup
フィールド：なし
静的メソッド：Init, Cascade, Tile
バーチャルメソッド：NewBackground, HandleEvent, TileError
説明：メニューバーとステータスラインの間のデスクトップを表す単純なグループ．

TDialog
上位オブジェクト：TWindow
フィールド：なし
静的メソッド：Init
バーチャルメソッド：HandleEvent, GetPalette, Valid
説明：ダイアログボックスを作るための単純なオブジェクト．

TDosStream
上位オブジェクト：TStream
フィールド：Handle
静的メソッド：Init
バーチャルメソッド：Done, GetPos, GetSize, Read, Seek, Truncate, Write
説明：バッファを用いないMS-DOSファイルストリームを構築するためのオブジェクト．

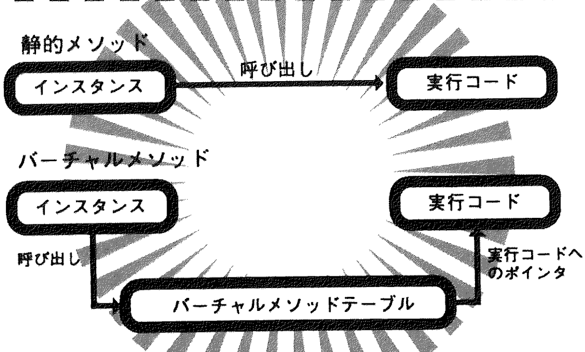
TEmsStream
上位オブジェクト：TStream
フィールド：Handle, PageCount, Size, Position
静的メソッド：Init,
バーチャルメソッド：Done, GetPos, GetSize, Read, Seek,

C++ & OOP 用語集
プリプロセッサ　ソースファイルの一部分をある条件で加工したり，読み飛ばしたり，他のソースファイルを読み込むなどの処理をする前処理プログラム．

(図2).

コンストラクタによって，オブジェクトのインスタンスにバーチャルメソッドテーブルへのポインタが渡されるわけである.

図2　バーチャルメソッド



オブジェクトの生成／消滅

Turbo Pascalのオブジェクトは，変数または型付き定数の宣言部で宣言されたり，オブジェクト型のポインタ変数に対して標準手続きNewが適用されたときに生成される（リスト6）.前者のように宣言されて生み出されるオブジェクトを静的インスタンスと呼び，これらはデータセグメントとスタック上に割り当てられる．後者のようにNewを用いて生み出されるオブジ

リスト5　コンストラクタ

```
var
    stest : StrField;                 { stestはStrFieldのインスタンス }
begin
    stest.Init(0, 0, 0, 'TEST');      { コンストラクタの呼び出し }
    ...
    stest.Done;
end;
```

リスト6　静的インスタンス／動的インスタンス

```
静的インスタンス
var
    stest : StrField;

動的インスタンス
var
    stest : ^StrField;

New(StrField);
```

Truncate, Write
説明：EMSメモリにストリームを構築するためのオブジェクト.

TFrame
上位オブジェクト：TView
フィールド：なし
静的メソッド：Init
バーチャルメソッド：Draw, GetPalette, HandleEvent, SetState
説明：ウインドウやダイアログボックスを囲む特殊なフレームオブジェクト.

TGroup
上位オブジェクト：TView
フィールド：Last, Current, Buffer, Phase
静的メソッド：Init, Load, Delete, ExecView, First, FirstThat, ForEach, GetSubViewPtr, Insert, InsertBefore, Lock, PutSubViewPtr, Redraw, SelectNext, Store, Unlock
バーチャルメソッド：Done, ChangeBounds, DataSize, Draw, EndModal, EventError, Execute, GetData, GetHelpCtx, HandleEvent, SetData, SetState, Valid
説明：Turbo Visionの核を形成するグループ.

THistory
上位オブジェクト：TView
フィールド：Link, HistoryID
静的メソッド：Init, Load, Store
バーチャルメソッド：Draw, GetPalette
説明：以前行った操作やデータ入力のリストを作成し，もう一度実行できるようにする.

THistoryViewer
上位オブジェクト：TListViewer
フィールド：HistoryId
静的メソッド：Init, HistoryWidth
バーチャルメソッド：GetPalette, GetText, HandleEvent
説明：ヒストリーシステム中で使用される.

THistoryWindow
上位オブジェクト：TWindow
フィールド：Viewer
静的メソッド：Init
バーチャルメソッド：GetPalette, GetSelection, InitViewer
説明：ヒストリーリストビュアーを保持するために使用される.

TInputLine
上位オブジェクト：TView
フィールド：Data, MaxLen, CurPos, FirstPos, SelStart, SelEnd
静的メソッド：Init, Load, SelectAll, Store

C++ & OOP 用語集
フレンド関数　クラスのプライベートデータに外部の関数からアクセスが必要なとき，friend宣言を行う．この宣言がされた関数をフレンド関数という.

ェクトを動的インスタンスと呼び，これらはヒープ上に割り当てられる．

New手続きは，動的インスタンスを格納するのに十分な領域をヒープ領域に割り当て，ポインタにそのアドレスをセットして返す．このとき，引数にコンストラクタをとることで，同時にインスタンスを初期化することができる（リスト6）．

この動的インスタンスをメモリ上から解放するメソッドが，デストラクタ（destructor）である．デストラクタは，バーチャルメソッドテーブルから解放すべきオブジェクトの大きさを取り出して，これをヒープ領域を解放する標準手続きであるDisposeに渡すことで正確にメモリを解放する．

Turbo Vision

Turbo Visionは，オーバーラッピングウインドウをもつアプリケーションを作成するためのフレームワークとなるクラスライブラリである．ただし，Turbo Pascalではクラスという言葉をほとんど用いず，オブジェクトという言葉を使用しているため，マニュアルではオブジェクト指向ライブラリと呼んでいる．

Turbo Visionで実現可能な機能には，次のようなものがある．

- リサイズ，オーバーラッピング可能な複数ウインドウ
- プルダウンメニュー
- マウスサポート
- ダイアログボックス
- カラーインストレーション
- ボタン，スクロールバー，入力ボックス，チェックボックス，ラジオボタン
- キー入力，マウスクリックの処理

Turbo Visionにおけるプログラミングは，Turbo Visionからオブジェクトを生成し，継承したメソッドをオーバーライドすることで変更を加え，機能を拡張していくというかたちで行われる．したがって，Turbo Visionを利用してプログラミングを行うためには，

- どのようなオブジェクトがあるのか
- それぞれのオブジェクトはどのようなフィールド，メソッドをもっているのか
- オブジェクトの関係（階層構造）
- フィールド，メソッドの継承関係

などを理解する必要がある．以下，Turbo Visionの内容について紹介していきたい．

Turbo Visionのオブジェクト階層

Turbo Visionのオブジェクト型には，基本オブジェクト，ビューオブジェクト，ミュートオブジェクトの3つがあり，さらにビューオブジェクトはTGroup以下のグループとそれ以外のターミナルビューに分かれている．

基本オブジェクトとは，他のオブジ

バーチャルメソッド：Done，DataSize，Draw，GetData，GetPalette，HandleEvent，SetData，SetState
説明：基本的な行入力の文字列エディタ．

TLabel
上位オブジェクト：TStaticText
フィールド：Link，Light
静的メソッド：Init，Load，Store
バーチャルメソッド：Draw，GetPalette，HandleEvent
説明：ビュー中の選択可能なテキストの一部．

TListBox
上位オブジェクト：TListViewer
フィールド：List
静的メソッド：Init，Load，Store
バーチャルメソッド：DataSize，GetData，GetText，NewList，SetData
説明：もっとも一般的なリストボックス．

TListViewer
上位オブジェクト：TView
フィールド：HScrollBar，VScrollBar，NumCols，TopItem，Focused，Range
静的メソッド：Init，Load，SetRange，Store
バーチャルメソッド：ChangeBounds，Draw，FocusItem，GetPalette，GetText，IsSelected，HandleEvent，SelectItem，SetState
説明：さまざまなリストビュアーの元になるもの．

TMenuBar
上位オブジェクト：TMenuView
フィールド：なし
静的メソッド：Init
バーチャルメソッド：Draw，GetItemRect
説明：メニューバーを表示する．

TMenuBox
上位オブジェクト：TMenuView
フィールド：なし
静的メソッド：Init
バーチャルメソッド：Draw，GetItemRect
説明：垂直方向のメニューボックスを表示する．

TMenuView
上位オブジェクト：TView
フィールド：ParentMenu，Menu，Current
静的メソッド：Init，Load，FindItem，HotKey，Store
バーチャルメソッド：Execute，GetItemRect，GetHelpCtx，

C++ & OOP 用語集
プロトタイプ　関数定義時に行う引数の型宣言のこと．ANSI Cではこのプロトタイプ宣言で引数の型をチェックする．

ェクトから使われたり，より複雑なオブジェクトの階層のベースとして使われるもので，TPoint，TRect，TObjectの3つがある．

ビューオブジェクトは，TViewから導出されるオブジェクトのことで，これらはすべて表示可能なオブジェクトである．

ミュートオブジェクトはビュー以外のオブジェクトのことで，これらが直接画面に表示されることはない．ミュートオブジェクトには，ストリーム，リソースファイル，コレクション，文字列リストなどがある．

図3にTurbo Visionに含まれるオブジェクトの階層構造を示す．

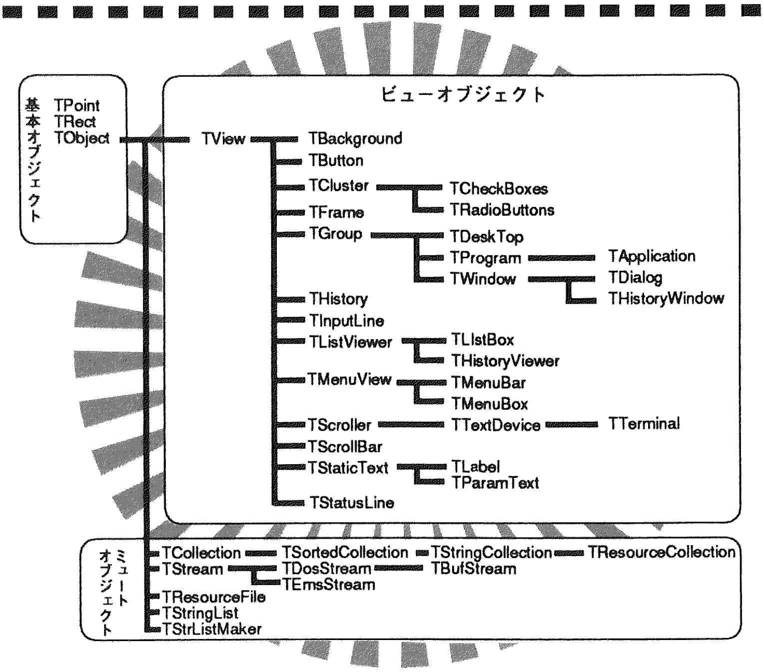
オブジェクトリファレンス

表1にTurbo Visionに含まれるオブジェクトの簡単なリファレンスを示す．フィールド，静的メソッド，バーチャルメソッドには，そのオブジェクトでオーバーライドしたものおよび追加されたもののみを記し，単に上位オブジェクトから継承しただけのものは省略した．

ビュー

Turbo Visionの中で中心的な役割を果たしているのは，なんといってもビューに属するオブジェクト群である．ここで，ビューオブジェクトがいかにウインドウシステムを実現しているのかについて見てみたい．

図3　Turbo Visionのオブジェクト階層構造



GetPalette, HandleEvent
説明：メニューバーとメニューボックスを導出する抽象オブジェクト．

TObject
上位オブジェクト：なし
フィールド：なし
静的メソッド：Init, Free
バーチャルメソッド：Done
説明：Turbo Visionオブジェクト階層の始点にあたる基本オブジェクト．

TParamText
上位オブジェクト：TStaticText
フィールド：ParamCount, ParamList
静的メソッド：Init, Load, Store
バーチャルメソッド：DataSize, GetText, SetData
説明：FormatStr手続きによって出力される文字列をパラメータとして渡す．

TPoint
上位オブジェクト：なし
フィールド：X, Y
静的メソッド：なし
バーチャルメソッド：なし
説明：単に画面の座標を指すオブジェクト．

TProgram
上位オブジェクト：TGroup
フィールド：なし
静的メソッド：Init, SetScreenMode, ValidView
バーチャルメソッド：Done, GetEvent, GetPalette, HandleEvent, Idle, InitDeskTop, InitMenuBar, InitScreen, InitStatusLine, OutOfMemory, PutEvent, Run
説明：Turbo Visionアプリケーションすべてのテンプレートを提供する．

TRadioButtons
上位オブジェクト：TCluster
フィールド：なし
静的メソッド：なし
バーチャルメソッド：Draw, Mark, MovedTo, Press, SetData
説明：ラジオボタンを作成する．

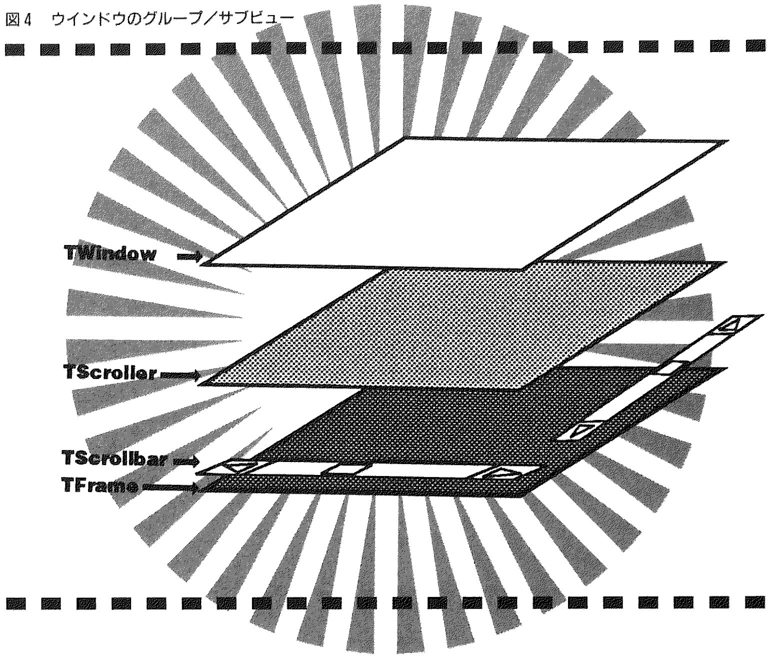
TRect
上位オブジェクト：なし
フィールド：A, B
静的メソッド：Assign, Copy, Move, Grow, Intersect, Union,

C++ & OOP 用語集
メンバアクセス制御　クラスのメンバはアクセス指定子によってアクセス属性を制御することができる．

グループ

ビューオブジェクトはグループとターミナルビューに分かれているが，グループとは要するに他のビューをその管理下におく空の箱のようなものである．実際のウインドウは，グループの上にいくつものサブビューが重なるような形で実現されている（図4）．

図4　ウインドウのグループ／サブビュー



Zオーダー

これらのサブビューをグループに組み入れることをインサートと呼び，サブビューがインサートされた順番のことをZオーダーと呼ぶ．また，画面上のユーザーから見て手前から奥へ向かって重なりあうサブビューの順番のこともZオーダーという．プログラム内でサブビューが呼び出されたときには，このZオーダーの順に画面に表示されることになる．

ビューツリー

ビュー同士の関係には，オブジェクト階層による継承の関係と，オブジェクト同士の所有関係の2つがある．後者の所有関係は，ビューツリーと呼ばれる図で示すことができ，あとで述べるイベントの発送はこのビューツリーに沿って行われる（図5）．

フォーカスチェーン

画面上に示されるウインドウの内，実際にアクティブになっているのは一部分のみであることが多い．このアクティブなウインドウの中で，選択されているサブビューのことをフォーカスビューと呼ぶ．TApplicationオブジェクトから，そこで選択されているサブビューを追って行くと最後にはフォーカスビューに達する．このビューのチェーンのことをフォーカスチェーンと呼び，イベントの発送がこのチェーンに沿って行われる．

モーダルビュー

Turbo Visionのビューには，動作

Contains, Equals, Empty
バーチャルメソッド：なし
説明：単に矩形を表す基本オブジェクト．

TResourceCollection
上位オブジェクト：TStringCollection
フィールド：なし
静的メソッド：なし
バーチャルメソッド：なし
説明：リソースのコレクションを実現するための抽象オブジェクト．

TResourceFile
上位オブジェクト：TObject
フィールド：Stream, Modefied
静的メソッド：Init, Count, Delete, Flush, Get, KeyAt, Put, SwitchTo
バーチャルメソッド：Done
説明：キー文字列をインデックスにできるストリームを実現する．

TScrollBar
上位オブジェクト：TView
フィールド：Value, Min, Max, PgStep, ArStep
静的メソッド：Init, Load, SetParams, SetRange, SetStep, SetValue, Store
バーチャルメソッド：Draw, GetPalette, HandleEvent, ScrollDraw, ScrollStep
説明：スクロールバーを表示する．

TScroller
上位オブジェクト：TView
フィールド：HScrollBar, VScrollBar, Delta, Limit
静的メソッド：Init, Load, ScrollTo, SetLimit, Store
バーチャルメソッド：ChangeBounds, GetPalette, HandleEvent, ScrollDraw, SetState
説明：スクローラを実現する．

TSortedCollection
上位オブジェクト：TCollection
フィールド：なし
静的メソッド：なし
バーチャルメソッド：Compare, IndexOf, Insert, KeyOf, Search
説明：重複することのないキーによってソートされているコレクションを実現する．

TStaticText
上位オブジェクト：TView
フィールド：Text

C++ & OOP 用語集
メンバ関数　クラス定義の中で宣言された関数のこと．そのクラス型と密接に結びついている．ほかのオブジェクト指向言語ではメソッドと呼ばれる．

モードを定義できるようになっており，この動作モードを定義したビューのことをモーダルビューと呼ぶ．

ビューを実体化しモーダルにした場合，そのビューおよびそれに属するサブビューだけがアクティブになる．つまり，ビューツリーの中で，モーダルビューとその下位にあるサブビュー以外は休止状態になる．

イベント駆動型プログラミング

Turbo Visionのプログラムは，イベント駆動型と呼ばれるものである．イベントとは，マウスのクリックやキーボードからの入力のことで，これらのイベントがプログラム内を送られていき，そのイベントに対応した正しいコードが実行される．

プログラマは，イベントの受け取り方や送り方などをプログラミングする必要はなく，単にあるイベントに対する処理方法のみを記述すればよいことになる．イベントの受け取り，発送などの処理はすべてTurbo Visionがやってくれる．

イベントが生じると，まず最初に現在のモーダルビューに送られる．ここからイベントが送られるルートには，ポジショナルイベント，フォーカスイベント，ブロードキャストイベントの3つがある．

ポジショナルイベント

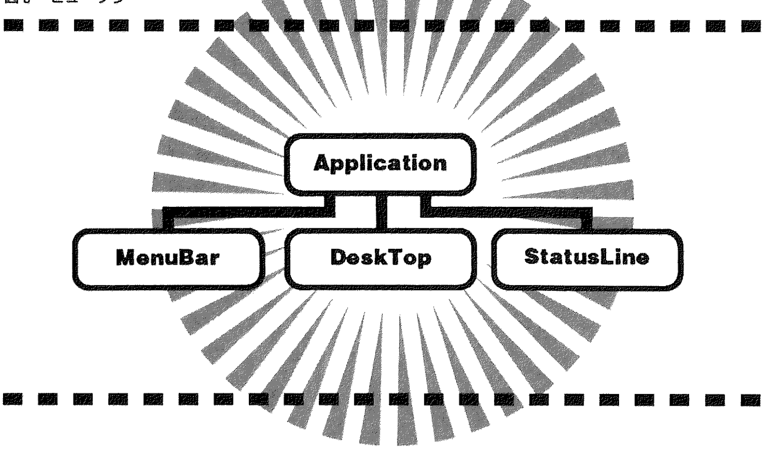
このイベントは，事実上マウスイベントのことである．

モーダルビューがポジショナルイベントを受け取ると，イベントの発生した場所にあるサブビューを見つけるまでZオーダーに沿ってサブビューを探索し，見つかるとそのビューにイベントを渡す．

フォーカスイベント

キー入力またはコマンドによるイベントで，フォーカスチェーンに沿って渡される．まずモーダルビューがイベントを受け取り，次にそのビューの管理下にあるサブビューにイベントを渡し，またそのサブビューというように次々にイベントが渡されていく．これがターミナルビューに達するまで続け

図5　ビューツリー



静的メソッド：Init, Load, Store
バーチャルメソッド：Done, Draw, GetPalette, GetText
説明：固定的なテキストを含むもっとも単純なビュー．

TStatusLine
上位オブジェクト：TView
フィールド：Items, Defs
静的メソッド：Init, Load, Store, Update
バーチャルメソッド：Done, Draw, GetPalette, HandleEvent, Hint
説明：ステータスラインを実現する．

TStream
上位オブジェクト：TObject
フィールド：Status, ErrorInfo
静的メソッド：CopyFrom, Get, Put, ReadStr, Reset, WriteStr
バーチャルメソッド：Error, Flush, GetPos, GetSize, Read, Seek, Truncate, Write
説明：ストレージデバイスとの間で入出力を行うための抽象オブジェクト．

TStringCollection
上位オブジェクト：TSortedCollection
フィールド：なし
静的メソッド：なし
バーチャルメソッド：Compare, FreeItem, GetItem, PutItem
説明：ASCII文字列のソートされたリストを組み込んだ単純なオブジェクト．

TStringList
上位オブジェクト：TObject
フィールド：なし
静的メソッド：Load, Get
バーチャルメソッド：Done
説明：ストリームに格納された文字列にアクセスする機能を提供する．

TStrListMaker
上位オブジェクト：TObject
フィールド：なし
静的メソッド：Init, Put, Store
バーチャルメソッド：Done
説明：TStringListで使用される文字列リストを生成するためのオブジェクト．

TTerminal
上位オブジェクト：TTextDevice
フィールド：BufSize, Buffer, QueFront, QueBack

C++ & OOP 用語集
予約語　識別子として使えないキーワード．ANSI Cでは32語が予約語になっている．

られ，最終的にはフォーカスビューがこのイベントを処理する．

ブロードキャストイベント

ブロードキャストイベントは送り先が決められていないため，すべてのサブビューにイベントが渡される．このイベントは，ビュー同士の通信に利用される．

このように，Turbo Visionにおけるプログラミングでは，プログラムの流れをイベントの流れとして捉えることがひとつの大きなポイントになる．イベントの発生→ビュー中でのイベントの発送→イベントを受け取ったオブジェクトによる処理の実行，といったプログラムの流れをごく自然に追えるようになれば，Turbo Visionプログラミングはほぼものになったといえるだろう．

あとは，各オブジェクトの機能に精通することだが，このときには別売されているTurbo Visionのソースコードを読んでみるのもひとつの方法かもしれない．

ウインドウプログラミングを考える

さて，駆け足でTurbo Pascal 6.0のオブジェクト指向とTurbo Visionについて紹介してきたがいかがだろうか．消化不良の感が強いかもしれないが，いかんせんTurbo Visionは複雑なライブラリであり，これだけの紙面スペースでそのすべてを紹介するのはとうてい不可能である．実際，Turbo Pascal 6.0のパッケージには，589ページにもおよぶ分厚いTURBO VISION GUIDEというマニュアルが付属している．本格的にTurbo Visionを活用しようとするならば，まずはこの分厚いマニュアルじっくりと読みこなすことだろう．

イベント駆動型プログラミングによる，イベントがビューツリーの中を送られていき，そのイベントにあったオブジェクトに達するとそこで処理が行われるという話を読んで，MS-Windowsにおけるプログラミングを思い出した方もいるかもしれない．実際，グラフィックベースかキャラクタベースかという違いを除けば，WindowsとTurbo Visionは，イベント駆動型というプログラミングの考え方の点でほとんど同じといえる．

今後，プログラマの好むと好まざるとに関わらず，この手のプログラミングが主流になっていくと考えられる．そろそろPascalプログラマも構造化というキーワードからオブジェクト指向というキーワードに頭を切り替えたほうがよさそうだ．しかし，Pascalプログラマには，Turbo VisionもあればTurbo Pascal for Windowsもある．ま，前途は洋々といったところだろう．

Turbo Pascal 6.0
価格：29,800円
Turbo Pascal 6.0 PROFESSIONAL
価格：59,800円
問い合わせ先：
（株）ボーランドジャパン
TEL 03-3486-1400

静的メソッド：Init, BufDec, BufInc, CalcWidth, CanInsert, NextLine, PrevLines, QueEmpty
バーチャルメソッド：Done, Draw, StrRead, StrWrite
説明：文字列をバッファ処理して読み書きする出力専用端末を実現する．

TTextDevice
上位オブジェクト：TScroller
フィールド：なし
静的メソッド：なし
バーチャルメソッド：StrRead, StrWrite
説明：スクロール可能なTTYタイプのテキストを表示するデバイスドライバ．

TView
上位オブジェクト：TObject
フィールド：Owner, Next, Origin, Size, Cursor, GrowMode, DragMode, HelpCtx, State, Options, EventMask
静的メソッド：Init, Load, BlockCursor, ClearEvent, CommandEnabled, DisableCommands, Dragview, DrawView, EnableCommands, EventAvail, Exposed, GetBounds, GetClipRect, GetColor, GetComands, GetExtent, GetPeerViewPtr, GetState, GrowTo, Hide, HideCursor, KeyEvent, Locate, MakeFirst, MakeGlobal, MakeLocal, MouseEvent, MouseInView, MoveTo, NextView, NormalCursor, Prev, PrevView, PutInFrontOf, PutPeerViewPtr, Select, SetBounds, SetCommands, SetCursor, Show, ShowCursor, Store, TopView, WriteBuf, WriteChar, WriteLine, WriteStr
バーチャルメソッド：Done, HandleEvent, CalcBounds, ChangeBounds, DataSize, Draw, EndModal, Execute, GetData, GetEvent, GetHelpCtx, GetPalette, PutEvent, SetData, SetState, SizeLimits, Valid
説明：下位オブジェクトのための基本フィールドとメソッドを提供する．

TWindow
上位オブジェクト：TGroup
フィールド：Flags, ZoomRect, Number, Palette, Frame, Title
静的メソッド：Init, Load, StandardScrollBar, Store
バーチャルメソッド：Done, Close, GetPalette, GetTitle, HandleEvent, InitFrame, SetState, SizeLimits, Zoom
説明：TFrameを1個，インテリアTScrollerを1個，TScrollBarを1ないし2個所有するグループ．

特集
はじめての
オブジェクト指向
プログラミング

もっと詳しくオブジェクト指向プログラミングを学びたい人に

オブジェクト指向プログラミング一般

オブジェクト指向入門
Bertrand Meyer著／二木厚吉監訳・酒匂寛・酒匂順子共訳／アスキー／4,980円
オブジェクト指向プログラミング言語Eiffelを使って，オブジェクト指向プログラミングに基づいたソフトウェア設計法を解説する．ソフトウェア工学の立場からの解説だけに，Eiffel言語以外の言語を使用した設計にも適用できる．

オブジェクト指向のプログラミングーソフトウェア開発技法の進化ー
B.J.コックス著／前川守監訳／トッパン／3,914円
直面するソフトウェア危機を打開する手段としての，オブジェクト指向プログラミングの必要性を説き，オブジェクト指向プログラミング言語Objective-Cによる問題解決法を解説する．

オブジェクト指向プログラミング
石塚圭樹著・横手靖彦監修／アスキー／2,880円
Cプログラマのためのオブジェクト指向プログラミング解説書．C，C++，Smalltalkの比較に加えて，ウィンドウシステムやHyperCardにも言及している．

C++

プログラミング言語C++
B.ストラウストラップ著／斎藤信男訳／トッパン／4,017円
C++の設計者であるAT&Tベル研究所のストラウストラップ自らが書き下したC++プログラミングの定本．AT&Tの1.2版に対応．

C++言語入門
S.C.デューハスト・K.T.スターク著／小山裕司監訳／アスキー／2,300円
基本的なプログラミング方法からオブジェクトの設計方法まで段階をふんで解説したC++言語の入門書．AT&Tの2.0版に準拠している．

C++ Cプログラマのための実践ガイド
S.ヘクマトプール著／岩谷宏訳／ソフトバンク／3,600円
中級者以上の人向けの具体的かつ実践的なC++プログラミング手法の解説書．

C++:オブジェクト指向プログラミング
R.S.ウィナー・L.J.ピンソン著／前川守訳／トッパン／3,200円
オブジェクト指向プログラミングのパラダイムとC++についての解説書．中級プログラマやC++・オブジェクト指向プログラミングを学ぶ学生向け．

C++プログラミング
門内淳・赤堀一郎著／ソフトバンク／2,680円
Cをある程度身につけた人を対象とした「C STEPUPシリーズ」の1冊．GUIDELINES C++ Ver.1.1およびVer.1.2，MIWA C++ Ver.1.2.3，ZORTECH C++ Ver.1.06に対応したC++の学習書．

Turbo C++ オブジェクト指向プログラミング入門
中山茂著／技報堂出版／2,369円
Turbo C++を用いたオブジェクト指向プログラミングの入門書．Turbo C++を使ってオブジェクト指向のグラフィックスライブラリを作成する．Turbo C++のインストールから統合開発環境IDEの使い方，CやC++の基本的な知識，C++の特徴の解説なども盛り込まれている．

C++初級プログラミング
石島辰太郎・小山裕司・大前昭博著／啓学出版／1,650円
サンプルプログラムによるC++の実

C++ & OOP用語集
列挙型　データのとりうる値をすべて列挙することにより定義されるデータ構造．

践的な入門書.

C++入門
足立高徳著／CQ出版／1,550円
例題プログラムを作る過程を通してC++の新しい機能を学ぶC++の入門書.

Smalltalk

Smalltalk-80―対話型プログラミング環境
A.ゴールドバーグ著／相磯秀夫監訳／オーム社／5,665円
米ゼロックス社で開発されたオブジェクト指向型環境Smalltalk-80のプログラミング編.

Smalltalk-80―言語詳解
A.ゴールドバーグ・D.ロブソン著／相磯秀夫監訳／オーム社／7,004円
Smalltalk-80の言語編.

基礎からのSmalltalk-80―オブジェクト指向のプログラミング
飯野昌一・及川一成・田制貴俊・川西真木著／工学社／3,000円
Smalltalk-80によるオブジェクト指向プログラミングを基礎から学ぶためのテキスト.

Smalltalk-80入門―オブジェクト指向プログラミングへのガイド
竹内郁雄監修・梅村恭司著／サイエンス社／2,060円
Smalltalk-80の基本概念と特長を，サンプルプログラムを交えて解説する.

Smalltalk
鈴木則久著／産業図書／2,060円
Smalltalk-80について，その使い方，言語の解説，パソコン上への実現方法などを解説する.

オブジェクト指向とSmalltalk
小林史典著／CQ出版／1,700円
PC-9801シリーズ上で動作する処理系Smalltalk/Vを使ってオブジェクト指向プログラミングの基礎と応用を解説する.

そのほかの言語

COMMON LISPオブジェクト指向（CLOS）
S.E.キーン著／稲葉浩人・河込和宏訳／トッパン／3,900円
Common Lispにオブジェクト指向の概念を取り込んだCLOS（Common Lisp Object System）の紹介と，これを基にしたオブジェクト指向プログラミングの基本概念の解説書.

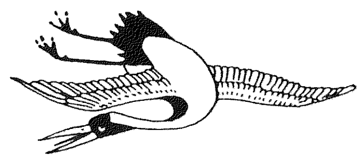
オブジェクト指向プログラミング for the Macintosh（上・下）
K.J.シュマッカー著／大谷和利監訳／ソフトバンク／上巻4,900円・下巻4,700円
Macintoshのユーザーインターフェースを構築するためのソフトウェアライブラリMacAPPの解説書. オブジェクト指向プログラミングの概念をPascalに移植したObject PascalからMacAPPを利用する方法や，Smalltalk-80によるアプリケーション開発などにもふれている.

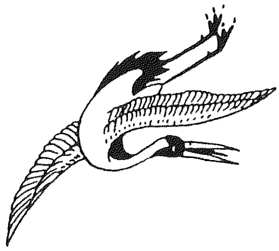
論理演算子　演算結果が記号論理学の規則に従い，オペランドのとりうる値のみからなる演算.

プログラム 遊 工房

谷口典生

八乃巻「サイクロイド/トロコイド/スピログラフ描画」プログラム







今回は，スピログラフを計算によって描くプログラムを作ってみました．スピログラフという名前には覚えがなくても「歯車状になったプラスチックの板を組み合わせてペンを差し込んで回すといろいろな模様が描けるやつ」といえば，誰でも一度は遊んだことのあるオモチャのことだと思い出してもらえることでしょう．
ワンポイントテクニックでは，暗黙のうちに行われる型変換について説明してみました．

プログラムの使い方

このプログラムはPC-9801シリーズ用で，Turbo C, QuickCおよびMS-Cに対応しています．リストはリスト1からリスト4の4つに分かれていますが，メインプログラムとなるのはリスト2，3，4の3つです．それぞれ「spiro1. c」「spiro2. c」「cycloid. c」というファイル名にして入力してください．リスト1はインクルードファイルとしてこれらのプログラムをコンパイルするときに使われるので，「graphsub. c」というファイル名にして同じディレクトリに入れておいてください．

今回のプログラムではグラフィック関数を使っているので，MS-CやQuickCでコンパイルするときにはグラフィックライブラリを指定します．MS-Cなら「CL spiro1. c /link graphics. lib」などとします．Turbo Cでは，実行時に. BGIファイルが必要となるので注意してください．

spiro1. exeとspiro2. exeは，

SPIRO1 m n p

SPIRO2 m n p

と3つの引数をとり，mとnは整数で大円と小円（内円と外円）の比（mとnは互いに素)を指定し，pは0から1の間の数で小円（外円）の中心からペンまでの距離を小円（外円）の半径との比で指定します．

たとえば，

SPIRO1 11 7 0.9

などとします．nの値を大きくすると，描画に時間がかかるので注意してください．

cycloid. exeはコマンドライン引数をとらないので，そのまま，

CYCLOID

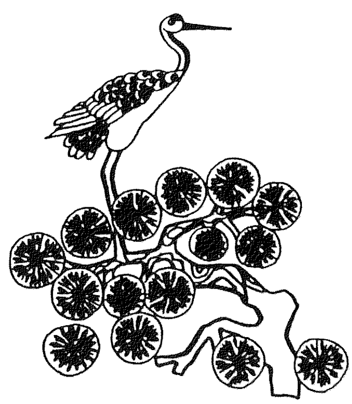
としてください(「サイクロイドとスピログラフ」の項を参照)．

リスト1

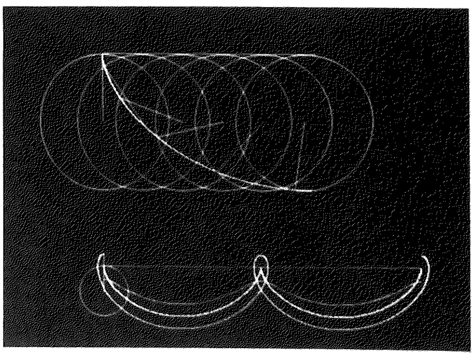
```
/*
 * graphsub.c             グラフィック関係のサブルーチン
 */

#ifdef __TURBOC__
/* Turbo C 用グラフィック初期化/復帰ルーチン */
#include <graphics.h>
void graphinit()                              /* グラフィックモード設定 */
{
        int graphdriver, graphmode;

        graphdriver = DETECT;                 /* 自動検出 */
        graphmode = PC98C8;                   /* 8色モード */
        initgraph(&graphdriver, &graphmode, "");
}
void graphdeinit()                            /* グラフィックモード復帰 */
{
        restorecrtmode();                     /* 画面モードを元に戻す */
        printf("¥x1b*");                      /* 画面クリア */
}
#else
/* MS-C, Quick C 用グラフィック初期化/復帰ルーチン */
#include <graph.h>
#define BLACK                   0
#define BLUE                    1
#define GREEN                   2
#define CYAN                    3
#define RED                     4
#define MAGENTA                 5
#define YELLOW                  6
#define WHITE                   7
#define setcolor                _setcolor
#define moveto(x,y)             _moveto(x,y)
#define lineto(x,y)             _lineto(x,y)
#define putpixel(x,y,c)         _setcolor(c);_setpixel(x,y)
#define circle(x,y,r)   _ellipse(_GBORDER,(x)-(r),(y)-(r),(x)+(r),(y)+(r))
void graphinit()                              /* グラフィックモード設定 */
{
        _setvideomode(_98RESSCOLOR);  /* グラフィック8色+テキストモード */
        printf("¥x1b[>5h");                   /* カーソル消去 */
}
void graphdeinit()                            /* グラフィックモード復帰 */
{
        _setvideomode(_DEFAULTMODE);  /* デフォルト(テキスト80行)モード */
        printf("¥x1b[>5l¥x1b[>1l");           /* カーソル表示、最下行復帰 */
}
#endif
```



サイクロイド



プログラムの説明

今回のプログラムは座標の計算と画面へのプロットだけの単純な内容のものなので，要点だけに絞って説明することにします．

❖リスト1(graphsub.c)

リスト2以降の3つのプログラムで使う共通部分です．Turbo CとMS-C (QuickC) ではグラフィック関数の扱いが異なるため，それぞれの処理系に応じたグラフィックの初期化および復帰ルーチンと関数名をあわせるための定義が含まれています．

❖リスト2(spiro1.c)

7行

リスト1 (graphsub. c) を#include文によって取り込みます．#include文はライブラリのヘッダファイルを取り込むのにも使われていますが（5，6行目），普通，ライブラリのヘッダファイル名は「<>」で囲み，この行のようにソースリストを取り込む場合などは「""」で囲みます．この違いは，コンパイラが指定されたファイルをどこのディレクトリから探すかの違いで，具体的な動作は処理系に依存します．たとえば，MS-Cは「""」で囲まれている場合はまずカレントディレクトリから捜し，「<>」で囲まれている場合は標準のインクルードディレクトリ（環境変数INCLUDEに設定されたディレクトリ）から探します．

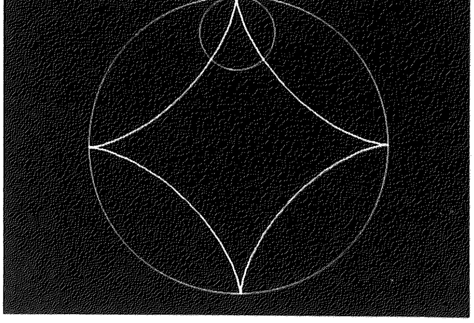
❖spiro1()

ペンの座標を計算して画面表示します．小円の中心と大円の中心を結ぶ線の傾きh1についてのループによって図形を描いていきます．見かけ上，小円は1周する間に転がる方向と逆の方向に1回転するため，ペンの位置の傾きh2はその分の補正を入れてh1に (m-n)/nをかけて計算しています．ペンが最初の位置に戻るまで小円を回転させればよいわけですが，ここではmとnの値が互いに素であると仮定してn周させています．本来はmとnが1以外の共役数をもつかどうかを判定して何周させれば十分かを計算するべきで

リスト2

```
 1: /*
 2:  * spirol.c               スピログラフ(spirograph) 内接型
 3:  */
 4:
 5: #include <math.h>
 6: #include <stdlib.h>
 7: #include "graphsub.c"
 8:
 9: #define XBASE   320                     /* 大円の中心 X座標 */
10: #define YBASE   200                     /* 大円の中心 Y座標 */
11: #define RADIUS1 198                     /* 大円の半径 */
12:
13: void spirol(int m, int n, double p)
14: {
15:         double h1,h2;           /* 小円及びペンの位置の傾き(ラジアン) */
16:         double r1,r2;           /* 大円、小円の半径 */
17:         double l1;              /* 大円の中心と小円の中心の距離 */
18:         double l2;              /* 小円の中心とペンの距離 */
19:         int x,y;                /* ペンのXY座標 */
20:
21:         r1 = RADIUS1;
22:         r2 = r1 * n / m;
23:         l1 = r1 - r2;
24:         l2 = r2 * p;
25:
26:         setcolor(BLUE);
27:         circle(XBASE, YBASE, (int)r1);                  /* 大円 */
28:         circle(XBASE, (int)(YBASE-r1+r2), (int)r2);     /* 小円の初期位置 */
29:         setcolor(GREEN);
30:         circle(XBASE, (int)(YBASE-r1+r2-l2), 4);        /* ペンの初期位置 */
31:
32:         moveto(XBASE, (int)(YBASE-l1-l2));
33:         for(h1=0; h1<3.14*2*n+0.04; h1+=0.04) {
34:                 h2 = h1 * (m-n) / n;
35:                 x = XBASE + l1*sin(h1) - l2*sin(h2);
36:                 y = YBASE - l1*cos(h1) - l2*cos(h2);
37:                 setcolor(WHITE);
38:                 lineto(x, y);
39:         }
40: }
41:
42: int main(int ac, char *av[])
43: {
44:         if(ac!=4)                               /* パラメータ数をチェック */
45:                 exit(1);
46:
47:         graphinit();                            /* 画面モードの初期化 */
48:
49:         spirol(atoi(av[1]), atoi(av[2]), atof(av[3]));
50:
51:         getch();
52:         graphdeinit();                          /* 画面モード復帰 */
53: }
54:
```

アステロイド



しょう．

❖リスト3(spiro2.c)

リスト2との本質的な違いは，2円の中心間の距離が（大円の半径）－（小円の半径）から（内円の半径）＋（外円の半径）に変わることと，転がる円が回転する方向とペンの位置が回転する方向が逆になることです．外円が内円の周りを回るためにおこる見かけ上の回転も外円の回転と同じ方向になるため，h1とh2の関係もh2＝h1＊（m＋n）/nとなります．

リスト3

```
/*
 * spiro2.c              スピログラフ(spirograph) 外接型
 */

#include <math.h>
#include <stdlib.h>
#include "graphsub.c"

#define XBASE   320                     /* 内円の中心 X座標 */
#define YBASE   200                     /* 内円の中心 Y座標 */
#define YSIZE   195.0                   /* 内円の半径と外円の直径の和 */

void spiro2(int m, int n, double p)
{
        double h1,h2;           /* 外円及びペンの位置の傾き(ラジアン) */
        double r1,r2;           /* 内円、外円の半径 */
        double l1;              /* 内円の中心と外円の中心の距離 */
        double l2;              /* 外円の中心とペンの距離 */
        int x,y;                /* ペンのXY座標 */

        r1 = YSIZE / (m+n+n) * m;
        r2 = YSIZE / (m+n+n) * n;
        l1 = r1 + r2;
        l2 = r2 * p;

        setcolor(BLUE);
        circle(XBASE, YBASE, (int)r1);                  /* 内円 */
        circle(XBASE, (int)(YBASE-r1-r2), (int)r2);     /* 外円の初期位置 */
        setcolor(GREEN);
        circle(XBASE, (int)(YBASE-r1-r2-l2), 4);        /* ペンの初期位置 */

        moveto(XBASE, (int)(YBASE-l1-l2));
        for(h1=0; h1<3.14*2*n+0.04; h1+=0.04) {
                h2 = h1 * (m+n) / n;
                x = XBASE + l1*sin(h1) + l2*sin(h2);
                y = YBASE - l1*cos(h1) - l2*cos(h2);
                setcolor(WHITE);
                lineto(x, y);
        }
}

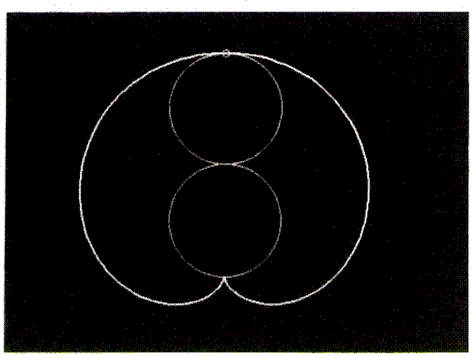
int main(int ac, char *av[])
{
        if(ac!=4)                               /* パラメータ数をチェック */
                exit(1);

        graphinit();                            /* 画面モードの初期化 */

        spiro2(atoi(av[1]), atoi(av[2]), atof(av[3]));

        getch();
        graphdeinit();                          /* 画面モード復帰 */
}

```



カルジオイド

サイクロイドとスピログラフ

リスト4（cycloid.c）のプログラムを実行してみてください．白く描かれた曲線は，ある円（青く描かれている円）が直線（赤い線）の上をすべらずにころがるときの円上の点の軌跡で，サイクロイドと呼ばれる曲線です．

画面上側の白い線は，円が半周だけするとき，初期位置で直線と接する円上の点が描くサイクロイドを示したものです．画面上方から下方に向かって一様な重力の場があるとして，白い線（＝サイクロイド）にそって質点が摩擦なしに右端まですべり落ちると仮定します．このとき，どの場所に質点を置いても，右端の点に到達するまでにかかる時間は変わらないのです．質点を左のほうの傾きが大きい部分に置けば加速はよいのですが，右端の点までの距離は長くなり，逆に右のほうに置けば距離は短くてもあまり加速しないというわけです．このような性質から，サイクロイドは「等時曲線」とも呼ばれます．

さらに，このサイクロイドの左端をA点，途中の点を任意にとってB点とすると，A点とB点を通る曲線にそって質点をすべらせる場合，所要時間が最短になるのは曲線がこのサイクロイドと一致するときになるのです．つまり，空間上のある点Aとそれより下にある点Bとを通る道で質点がすべり落

リスト4

```
/*
 * cycloid.c              サイクロイド(及びトロコイド)を描く
 */

#include <math.h>
#include <stdlib.h>
#include "graphsub.c"

#define XBASE   100             /* X座標ベース */
#define YBASE   100             /* cycloid1() で描く図形のY座標ベース */
#define YBASE2  300             /* cycloid2() で描く図形のY座標ベース */
#define RADIUS  80              /* cycloid1() の円の半径 */
#define RADIUS2 30              /* cycloid2() の円の半径 */
#define RADIUS3 15              /* トロコイド1の極と転円中心の距離 */
#define RADIUS4 45              /* トロコイド2の極と転円中心の距離 */

void cycloid1()                 /* サイクロイドの表示 */
{
        double h1;              /* 転円の回転角 */
        int x,y;                /* ペンのXY座標 */
        int n;

        setcolor(RED);                                  /* 底線を描く */
        moveto(XBASE, YBASE-RADIUS-1);
        lineto(XBASE+(int)(RADIUS*3.14), YBASE-RADIUS-1);

        for(h1=0,n=0; h1<3.14; h1+=0.01,n--) {
                x = XBASE + (int)(RADIUS*h1 - RADIUS*sin(h1));
                y = YBASE - (int)(RADIUS*cos(h1));
                putpixel(x, y, WHITE);
                if(n==0) {                      /* 途中で転円の状態を描く */
                        n = 60;
                        setcolor(BLUE);
                        circle(XBASE+(int)(RADIUS*h1), YBASE, RADIUS);
                        moveto(XBASE+(int)(RADIUS*h1), YBASE);
                        lineto(x, y);
                }
        }
}

void cycloid2()                 /* サイクロイドとトロコイドの表示 */
{
        double h1;
        int x,y;

        setcolor(RED);                          /* 底線を描く */
        moveto(XBASE, YBASE2-RADIUS2);
        lineto(XBASE+(int)(RADIUS2*3.14*4), YBASE2-RADIUS2);
        setcolor(BLUE);                         /* 転円の初期位置を描く */
        circle(XBASE, YBASE2, RADIUS2);
        moveto(XBASE, YBASE2);
        lineto(XBASE, YBASE2-RADIUS4);

        for(h1=0; h1<3.14*4; h1+=0.02) {
                x = XBASE + (int)(RADIUS2*h1 - RADIUS2*sin(h1));
                y = YBASE2 - (int)(RADIUS2*cos(h1));
                putpixel(x, y, WHITE);                  /* サイクロイド */
                x = XBASE + (int)(RADIUS2*h1 - RADIUS3*sin(h1));
                y = YBASE2 - (int)(RADIUS3*cos(h1));
                putpixel(x, y, GREEN);                  /* トロコイド1 */
                x = XBASE + (int)(RADIUS2*h1 - RADIUS4*sin(h1));
                y = YBASE2 - (int)(RADIUS4*cos(h1));
                putpixel(x, y, CYAN);                   /* トロコイド2 */
        }
}

void main()
{
        graphinit();                            /* 画面モードの初期化 */

        cycloid1();
        cycloid2();

        getch();
        graphdeinit();                          /* 画面モード復帰 */
}
```

リスト5

```
main() {
        char c;
        int i;

        c = 200;
        i = c;
        printf("i=%d¥n", i);
}
```

ちるのにもっとも時間がかからない道を得るには，A点を通る水平線を底線(円が接しながらころがる直線を底線という）とするサイクロイドでB点を通るものを選べばよいのです．この性質から，サイクロイドは「最短降下線」とか「最速降下線」と呼ばれることもあります．

だからどうしたといわれると困りますが，サイクロイドがこのような性質をもっていることを考えながら画面に描かれる曲線を見ていると，なんとなくありがたい気がするのではないでしょうか．

画面下側にはサイクロイドの白い線とともに水色の線と緑の線も描かれていますが，これはサイクロイドが円周上の点の軌跡であるのに対して，円の内側の点や外側の点（仮想上の点ということになります）の軌跡で，トロコイドと呼ばれる曲線です．底線（赤い線）が直線でなく円周であれば，転がる円は底線となる円に外接または内接しながら転がることになります．外接する場合のサイクロイド(トロコイド)を外サイクロイド(外トロコイド)，内接する場合は内サイクロイド（内トロコイド）と呼び，これこそスピログラフの描く曲線そのものです．

そのなかで有名な名前がついている曲線を２つ紹介しておきましょう．

アステロイド

底線となる円と転がる円の半径の比が４：１となる場合の内サイクロイド(SPIRO1 4 1 1)．直交座標に関する方程式，X^(2/3)+Y^(2/3)=a^(2/3)であらわされる図形．

カルジオイド（心臓形）

底線となる円と転がる円の半径が同じになる場合の外サイクロイド(SPIRO2 1 1 1)．方程式，(X^2+Y^2-aY)^2=a^2(X^2+Y^2)であらわ

される図形．

ワンポイントテクニック

暗黙のうちに行われる型変換に注意しよう

C言語では，式の中に異なる型の被演算数があると同じ型に変換して計算されます．たとえばint型の変数 i と float型の変数 f があった場合，(i+f) は i がfloat型に変換されて計算され，結果はfloat型となります．また，ある型の式が異なる型の変数へ代入されたときも自動的に型の変換が行われます．関数に渡す引数についても同様です．

多くの場合こういった型変換は暗黙のうちに行われますが，計算式の中でどのような型変換が起こっているかをきちんと把握せずにプログラムを書いていると，思わぬバグに悩まされることがあります．

リスト５のプログラムを実行すると，どのような結果が得られるでしょうか．MS-C (QuickC) やTurbo C でコンパイルして実行すると「i=-56」と表示されるはずです．なぜそうなるのか，順を追って見てみましょう．

まず，char型の変数 c に200という(int型の)定数を代入していますが，ここで型変換が行われます．char型は(処理系に依存します) MS-CやTurbo Cではsignedなので，(-128)から127までの値をとり，200というint型の数は(-56)という値として c に代入されます．次の「i = c;」では，そのまま (-56) がint型の (-56) に拡張されて i に代入され，結果として i の値は (-56) になるのです．

char型の変数を１バイトであらわせる範囲の数 (０から255) として利用したいときもあります．このときは，unsigned char型を使えば問題ありません．リスト５のプログラムも c を unsigned char型として宣言すれば，結果は「i=200」となります．もしくは「i = c & 0xff;」のようにint型の0xff と ANDをとってやるか，「i = (unsigned char) c;」といったんunsigned char型にキャストしてから代入するという手もあります．

リスト６を見てください．このプログラムではどのような結果が得られるでしょうか．まず最初の a は右辺がint型として計算されるので，まず10を３で割った時点で切り捨てが起こって３になり，次に３をかけて９という結果になります．

このようなint型での除算の切り捨てによる計算の誤差をできるだけ少なくするため，除算をあとまわしにするという手を使うことがあります．b は先に３をかけてから３で割っているので，結果は10になります．除算と乗算の優先順位が同じで，左から計算が行われていることがよくわかります．

次の行では10を10.0と書くことによってfloat型の定数とし，右辺の式をfloat型で計算させることによって最終的に c に (int型に変換されて) 代入される値は10となります．異なる型の被演算数が現れると，より情報の多い型 (char型よりint型，int型よりfloat型，float型よりdouble型といったように) へと格上げされて計算され，結果もその型となることに注意してください．

もっと明示的に10にfloat型をキャストしてやることによっても同じ結果が得られ，d の値は10になります．

次の行のように最後の３だけをfloat型にしても，最初の除算がint型で行われてから乗算がfloat型で行われるので，結果としては e の値は９になり，最初の a の計算と同じ結果になってしまいます．

勘違いしないように気をつけたいのが f のような場合で，代入される側の変数がfloat型であっても，右辺はint型で計算されたあとに結果の９がfloat型に変換して代入されるだけなので，f の値は9.00になります．

このように暗黙的な型の変換が行われるのはC言語の便利なところでもあり，危険なところでもあります．異なる型を混ぜた式などを使うときは，予期しない型変換が起こっていないか十分気をつけるようにしましょう．

リスト６

```
main()
{
        int a,b,c,d,e;
        float f;

        a = 10 / 3 * 3;
        b = 10 * 3 / 3;
        c = 10.0 / 3 * 3;
        d = (float)10 / 3 * 3;
        e = 10 / 3 * 3.0;
        f = 10 / 3 * 3;
        printf("a=%d,b=%d,c=%d,d=%d,e=%d,f=%f¥n",a,b,c,d,e,f);
}
```

パソコン博物館 VIII

オールインワン・パソコン
OKI if800

沖電気工業　OKI if800

井戸禎光

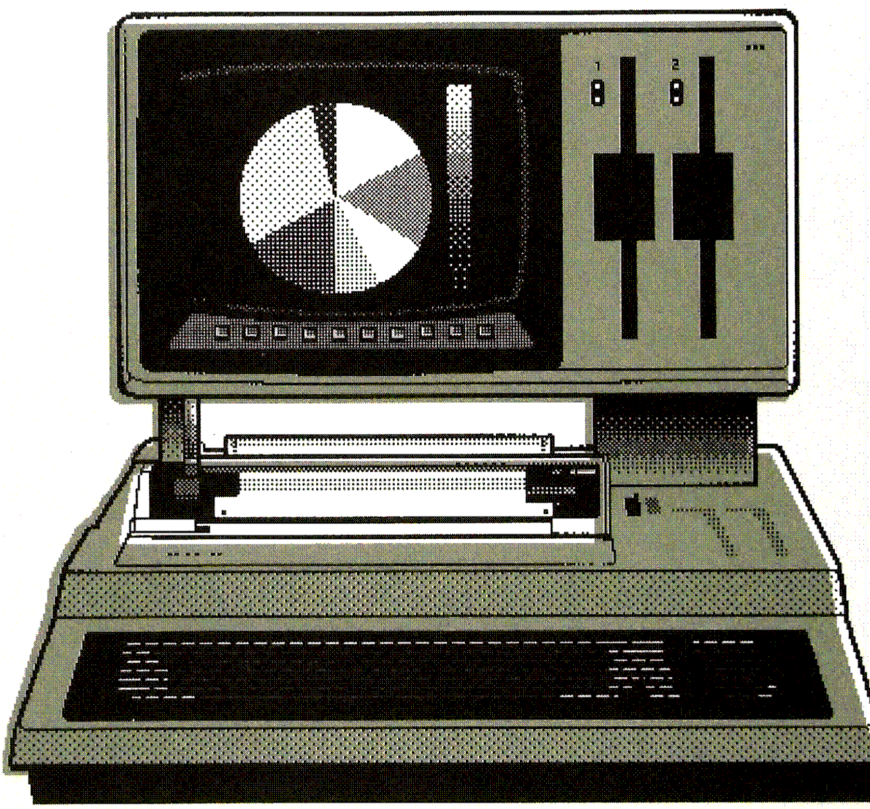


Illustration
TOSHIHITO ITO

今回は，沖電気が1980年5月に発表したif800について考えてみたい．これはプリンタとCRTのついたオールインワンの機械であった．はじめに当初からのコンセプトとして，沖電気のもっとも強いところを利用していくという方針のもとに，プリンタありきというところから出発し，全体の基本設計がされている．

if800という名前の根拠は，インテリジェント・ファシリティであるといわれているが，後ろの800についてはわからない．

if800のもともとの誕生のいきさつを探るとおもしろい．1970年代は米国ではベンチャービジネスがパソコンを作り，日本では大手電気会社がボードコンピュータのキットを作りはじめたころである．このころは標準といってもMITS社のS-100バスがあったくらいで，誰もが思い思いのものを作ってみて，その成果を世の中に問うといった状態であって，設計者の個性が現れた作品が多かった．

このような状況において沖電気は，B4サイズのボードにZ80を入れて，キーボードを取りつ

け，それに家庭用のTVとカセットレコーダをつなげば，大型コンピュータの仕事の一部ができるのではないかと考えた．

そうこうするうちに，マイコンショウに出そうということになり，PD-80と名づけて1979年マイコンショウに参考出展した．それには，当時各種あったBASICのうちから“ポテトBASIC”を選び搭載した．

一方，会社全体としては，本格的にパソコンビジネスに参入したいというムードが広がっており，当時あったミニコングループの中にSBU（Strategic Business Unit）が作られた．SBUは，製品別に技術，営業，保守など，ビジネスをはじめていく上で必要なすべての機能をもたせた組織の作り方である．その1つがワープロのグループであり，このグループは，レターメイトという価格は150万円のものを出した．そして，もう1つがこのif800を作ったグループであった．

さて，正式のSBUグループの方針は，“プリンタつきのオールインワンタイプ”というコンセプトになった．すなわちCPU本体にプリンタ，ディスプレイ，FDD，RS-232Cが標準でついたものとなる．そうなると価格レンジが100から150万円（実際は148万円）となり，どうしてもビジネスユースとなる．もっとも，これはモデル20のことであり，ペリフェラルを除いて本体だけにしたモデル10もあったのであるが，商品イメージがオールインワンを正面に出していたので，あまり大きな影響力を及ぼさなかった．if800は売り出すとすぐに反響があり，2年間で海外に4000台，国内に10,000台，合計14,000台を売った．

本格的にビジネスユースに使われてみると，10インチのプリンタでなく16インチのプリンタが欲しい人とか，8ビットで64KBではCP/M，OKIBASICを入れるとメモリ容量がたりなくなるといった要求が出てきた．そこでCPUにZ80Bを使い，CRTを大きくして640×400ドットにし，プリンタは16ドットの1回打ちとし（モデル20ではじめて漢字をプリンタで打たせたのであるが8ドットの3回打ちであった），ROMをプリンタにも入れてドットでなくコード信号を送ればよいようにしてプリンタの高速化をはかり（本邦初？），バンク切り替えによって256KB実装を可能にした．FDDもIBM社の1MBの物理フォーマットに合わせて，汎用機でも読めるデータ構造にした．これがif800モデル30である．

このようなモデル20/30の時代は日本語の使える大容量機としてISV（独立系ソフトウェア会社）のメインターゲットはif800であり，1982年5月のビジネスショウには日本語ワープロとか，表計算，データベースなど，数々のビジネスパッケージが出展されている．

if800を考えるとき，沖電気の潜在的技術力の高さに驚くのである．すなわち，研究所の中にふと思いついてパソコンを作ってしまえるような人がいる．また，必要な予算とか測定器とかの設備がすでに整っている．SBUグループができたのはラッキーであったが，その次にビジネスとして大成させられるような，パソコンの本質を理解できる人がすぐに集まった．

もっとも沖電気は，1960年代のICが出る前に，当時の大型汎用機でトップ企業であり，OKITACは計算機のことを意味していると思っていたのは筆者だけではあるまい．また，その後TTLが盛んになったころ，はやくも500シリーズ・5000シリーズのCMOSに力を入れていて，多くの時計用ICに発展していった．

このような長い歴史の中に育まれた総合力が，組織の中に実在する技術力なのではないか．

●if800モデル20のI/Oスロット信号

端子	信号名	方向	端子	信号名	方向
A 1	+5V	出 力	B 1	+5V	出 力
A 2	+5V	出 力	B 2	+5V	出 力
A 3	+12V	出 力	B 3	+12V	出 力
A 4	GND		B 4	GND	
A 5	(N)EXT CLK	出 力	B 5	(N)RESET	出 力
A 6	空き		B 6	CLK_2	出 力
A 7	空き		B 7	(メーカリザーブ)	
A 8	(メーカリザーブ)		B 8	(N)EXT IOREQ	
A 9	(N)EXT WAIT	出 力	B 9	(N)EXT WR	
A10	(メーカリザーブ)		B10	(メーカリザーブ)	
A11	GND		B11	GND	
A12	AEN		B12	(メーカリザーブ)	
A13	(N)EXT WAIT	入 力	B13	(N)EXT INT	入 力
A14	(メーカリザーブ)		B14	(N)EXT DATAIN	入 力
A15	GND		B15	GNC	
A16	EXT A_{15}	入出力	B16	EXT A_{14}	入出力
A17	EXT A_{13}	入出力	B17	EXT A_{12}	入出力
A18	EXT A_{11}	入出力	B18	EXT A_{10}	入出力
A19	EXT A_9	入出力	B19	EXT A_8	入出力
A20	EXT A_7	入出力	B20	EXT A_6	入出力
A21	EXT A_5	入出力	B21	EXT A_4	入出力
A22	EXT A_3	入出力	B22	EXT A_2	入出力
A23	EXT A_1	入出力	B23	EXT A_0	入出力
A24	GND		B24	GND	
A25	EXT D_7	入出力	B25	EXT D_6	入出力
A26	EXT D_5	入出力	B26	EXT D_4	入出力
A27	EXT D_3	入出力	B27	EXT D_2	入出力
A28	EXT D_1	入出力	B28	EXT D_0	入出力
A29	−12V	出 力	B29	−12V	出 力
A30	−5V	出 力	B30	−5V	出 力

祐安重夫

プログラマの道具学

第8回 日本語ワードプロセッサ

8年前に1台目のPC-9801を購入したのは，もちろんソフトウェア開発のためだったが，それと同時に日本語ワードプロセッサとして利用したいという理由もあった．その前の年から文章をかくことを仕事として本格的におこなうようになっていたし，ソフトウェアのドキュメントの作成にも必要を感じていたところだった．うらがえせば，字が下手だということでもあったのだが．

当時はちょうど初期の日本語ワープロ・ソフトがではじめたところで，全角文字と半角文字が混在してつかえるといった現在ではあたりまえのことさえもが，製品の特徴として宣伝されるような時代だった．仮名漢字変換についていえば，やっと熟語変換が実用化されてきたころである．

最近のパーソナル・コンピュータ上のワープロ・ソフトは，ほとんどがMS-DOS上で動作しているが，当時はBasicの上か独自の環境下でうごくものしかなかった．2年ほどして〔一太郎〕が登場し，MS-DOS上で動作するという，現在でも標準となっているような環境が実現されるようになる．

このころにはプログラミングも，MS-DOS上でC言語をつかっておこなうのが標準的な方法となり，ソフトウェアの開発からドキュメントの作成まで，すべてをおなし環境で実行できるようになってきた．仮名漢字変換の部分もフロント・エンド・プロセッサとして独立するようになり，プログラムのなかに日本語のコメントを，ワープロとまったくおなじ操作方法でいれることも可能になった．

ここでネックになったのは，プログラムの入力につかうエディタと，ドキュメントの作成につかうワードプロセッサという，機能的には類似しているが，操作方法のまったくちがう二つのシステムをつかいわけなければならないことである．日本語ワープロも広義のテキスト・エディタであり，MS-DOS上で動作するものはMS-DOSのテキスト・ファイルを出力することができるので，ひとつの解決方法としてプログラムの入力にも日本語ワープロをつかうことがかんがえられた．しかし，エディタとしての機能からみるかぎりは，日本語ワープロの操作性はけっしてよいとはいえず，またスピードもおそかった．

逆に高機能でこまわりのきくテキスト・エディタを，ドキュメントの作成にも利用できたら，仕事の効率が大幅に向上するはずであ

Illustration
Mikio Sasaki

る．日本語ワープロのように画面上で文書のできあがりのイメージを完全に把握することはできないかもしれないが，つくられたファイルを別のプログラムで処理してやることで，最終的には日本語ワープロと同等の出力がえられればそれでいいのである．

このようなかんがえかたは特殊なものではなく，現在では最終的な出力だけに日本語ワードプロセッサを使用し，文章の入力には高機能のテキスト・エディタをつかうユーザが，かなりふえているようだ．

Unixには以前からnroffとかtroffとよばれるコマンドがあって，文書の入力と整形作業を分離するという発想で英文の文書作成をおこなっていた．エディタではできあがりのフォーマットをあまり意識せずに，最低限の約束事だけをまもりながら適当に文書を入力していき，nroffやtroffというプログラムで処理してやると，標準的な英文文書の書式にしたがって，きれいに清書されたプリント・アウトがえられるのである．もちろん標準以外の特別な書式が必要だったら，そのための指示を文書の中にいれてやることが可能である．

この方法の利点は，文書のできあがりのイメージという，本来なら編集や組版の専門家がかんがえるべき問題を，文書を作成する人間が意識する必要がないという点である．たとえば日本語ワープロで表をつくる場合，自分で全体のレイアウトを設計し，その中に項目をうめこんでいくという作業をおこなわなければならない．nroffやtroffをつかう場合は，表組を指示する命令を文書の中にいれて，あとは項目をならべておくだけである．表のレイアウトは各項目のおおきさにもとづいて，最適なものを自動的にプログラムが決定してくれる．もちろん特殊なレイアウトが必要なら，そのための指示を文書にいれておくことで対応できる．

日本語ワープロのような方法は，一見すると自由度がたかくわかりやすいようだが，じつは文書を作成する人間によけいな負担をかけているともいえるのである．もともとの目的はうつくしいできあがりの文書を作成することであり，そのために不必要な作業をしいられる必要はまったくない．

その後ソフトウェアの開発に，日本語のつかえるUnixを利用するようになったのを機会に，日本語対応のnroffのサブセットのようなものを自作し，それを利用するようになった．結局今では，日本語ワープロはまったくつかっていない．

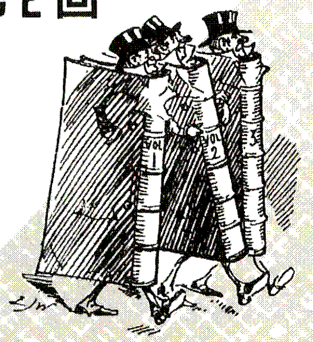
日本語ワードプロセッサとともに発達してきたものに，仮名漢字変換をおこなうフロント・エンド・プロセッサがある．VJEとかATOKとかいったものが有名であり，熟語変換からはじまって文節変換，連文節変換，文章一括変換，自動変換といった具合に，あたらしい機能が追加されてきているのだが，私自身は熟語変換かせいぜい文節変換のレベルにおさえて使用するようにしている．この文章をみてもわかるように，私の文体は比較的ひらがなのおおいものであり，連文節や一括変換といったものでは変換されすぎてしまうのである．

紙面構成やレイアウトの指定といったものでは，印刷文化の歴史の中でみがきあげられてきた方法を積極的にとりいれるべきであり，それを文章を作成する素人にまかせるという従来の日本語ワープロのやりかたには，かなり無理がある．しかし文体は個人の美意識の問題であり，それがフロント・エンド・プロセッサによってねじまげられてしまうことは，非常に不本意なことであり，のぞましくない．そういった意味では，現在の日本語ワープロのありかたには，おおきな問題があるとおもうのである．

IC千夜一夜

第2回

4ビットマイクロプロセッサの登場



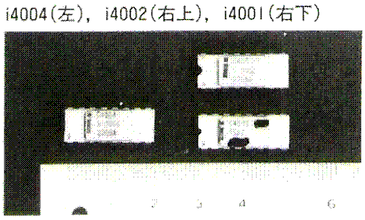
―4004と4040―

虎岩 登

今回は，史上初のマイクロプロセッサとされている4004とその後継者である4040を見てみましょう．4004が発表されたのは1971年のことなので，もう20年も前のことになります．4040が発表されたのは3年後の1974年です．

初のマイクロプロセッサ4004は，写真のとおり本当に小さなプロセッサで，内部のトランジスタ数も約2,200個でした．現在では100万をこえるトランジスタをもつICが存在することを考えると，小規模なプロセッサといえるかもしれませんが，当時は立派なLSIでした．

i4004(左)，i4002(右上)，i4001(右下)



ハードウェアアーキテクチャ

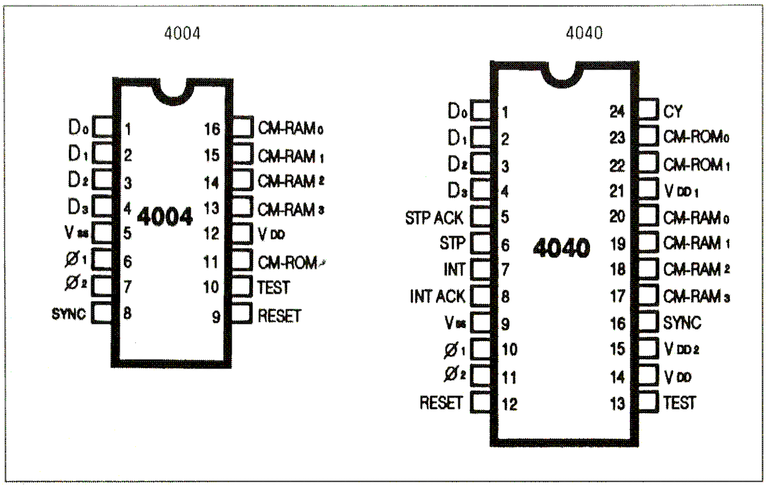
ピン配置から見た特徴

図1は，4004と4040のピン配置です．4004は16本，4040は24本の端子があります．4040では機能が増えた分だけ，ピン数が増えていますが，入出力タイミングなどはほとんど変更ありませんでした．4004用の周辺ICもそのまま流用できるようになっています．つまりハードウェア面では上位コンパチブルだったわけです．命令セットも14種類増えているだけで，あとはまったく同じです．後年，8080に対してZ-80とか8085というCPUが開発されていますが，同じような考え方はこの時代から続いているようです．

まずは，4004と4040の共通の仕様について見ていきましょう．プロセスは，p-MOS技術を使っています．したがって，電源は15V必要ですが，入出力電圧をTTLと同じにするために+5Vと-10Vの2電源で使うのが一般的です．ϕ1とϕ2の2相クロックが必要で最大クロック周波数は750KHzです．p-MOSですから，クロック周波数はあまり上げられないのです．命令には1バイト命令と2バイト命令の2種類があり，1バイト命令は約10.8μs，2バイト命令は21.6μsで実行されます．1バイト分の命令を処理するのに，8クロックが必要です．1命令で実行できる内容が全然違うので，ほかのプロセッサと比較するのは無意味なのですが，0.1MIPS以下の能力というわけです．

なぜ，こんなに遅いのでしょうか．そのヒントはピン配置にあります．D0-D3の4本のデータバスらしい端子がありますが，アドレスバスに相当する

図1　4004・4040のピン配置



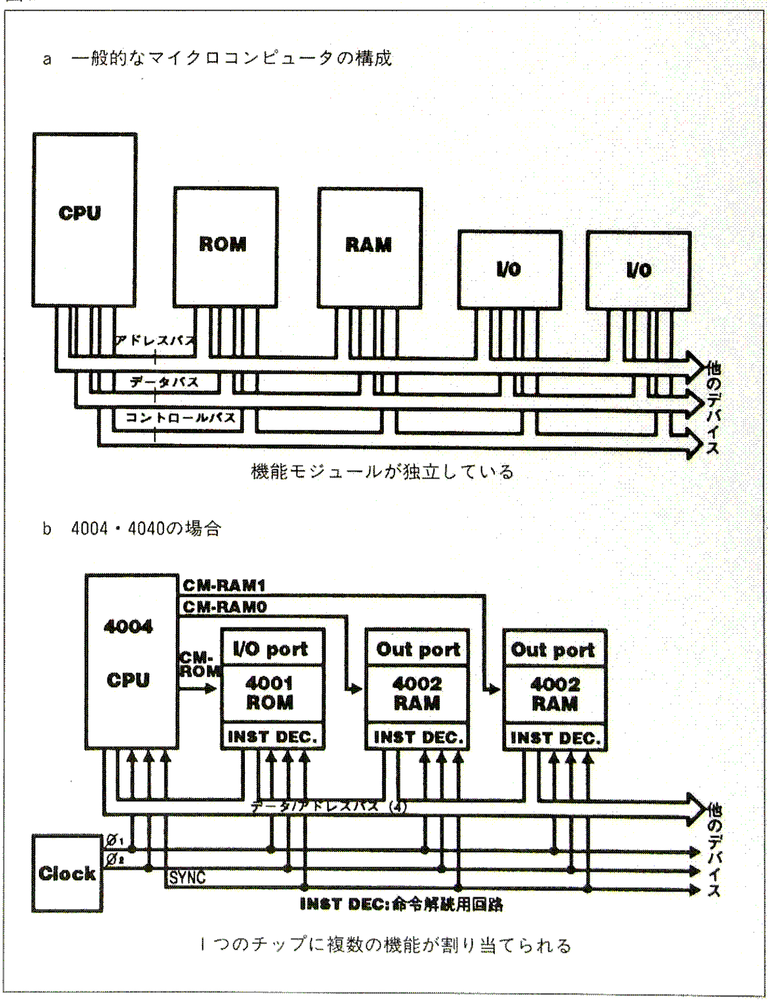
端子が見あたりません．CM-RAMとかCM-ROMという端子がアドレス指定のように見えますが，この端子は実際にはメモリ空間を増やすための一種のバンク切り替えの端子であって，アドレスバスではありません．D0-D3の4本の端子で，アドレスと命令，データのすべてを入出力するのです．アドレスは12ビットで指定するので各クロックごと3回に分けて出力され，それから8ビットの命令を2回に分けて読み込み，その後で3クロックかけて命令を実行します．

データを読み込むタイミングや，書き込むタイミングを指定する端子がないこともユニークな点です．周辺のメモリやI/Oポートは，SYNCという，アドレスを出しはじめる直前に立ち上がる信号とクロックを見ていて，CPUからアドレスを受け取ったりするだけでなく，同時にメモリから出力される命令を監視して命令実行時にCPUが何を行うか予測する必要がありました．CPUが入出力命令を読み込んだら，該当するI/Oデバイスはデータを受け取るタイミングをクロックから見定めなければなりません．

周辺デバイスにも，CPUと似たような命令の解釈をする回路が必要なのです．4004や4040のアーキテクチャでは，すべての周辺デバイスはCPUのコプロセッサであるといい切ってもいいかもしれません．

このような複雑なタイミングが要求される回路をいちいち作らなければならないとしたらたいへんですが，インテル社からはその回路を含んだICが提供されています．その代表的なものは，4001という型番の1KBのROMと，4002という80×4ビットのRAMチップです．これらのICには，4ビットの入出力ポートがいっしょに内蔵されているのも変わっています（4002は出力のみ可能）．一般のマイクロコンピュータのハードウェアから見たアーキテクチャは，図2aのように単機能のユニットが多数本のデータ・アドレスバスとコントロールバスで結ばれている

図2



かたちになっていますが，4004や4040のアーキテクチャでは図2bのようにデータ・アドレス共用バスとクロックとSYNC信号，さらにバンク切り替え用の線で，命令解釈，メモリ，入出力ポートが渾然一体となったICを結合しているかたちになります．

4004や4040の時代には，プログラム方式によってさまざまな目的への応用が考えられていたとはいっても，まだ特定の顧客を中心に販売することしか考えられなかった雰囲気がうかがえます．あくまで，インテルの提供する枠内の部品だけで主要部が構成されることを前提にしていたようです．

ハードウェア面で4004から4040になって変化した部分ですが，最大の違いは割り込みが可能になったことです．あとはバンク切り替えの端子が増えた分，利用可能なメモリ容量が増えたということ，強制的にCPUを停止させる端子が付いたことです．

ソフトウェアアーキテクチャ

レジスタ構成と命令セット

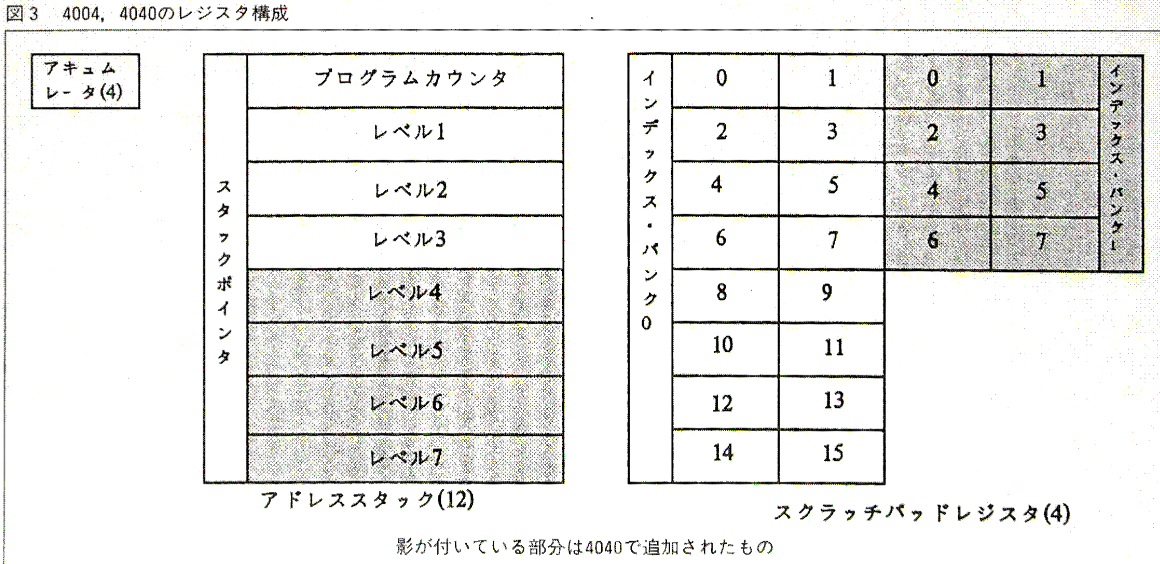
次に，ソフトウェア面から4004のアーキテクチャを見てみましょう．図3は4004のレジスタセットです．アキュムレータに加えて，インデックスレジスタとも呼ばれるスクラッチパッドレジスタが16個あります．スクラッチパッドレジスタというのは，データを一時的に入れておくためのレジスタで，加算や減算はアキュムレータとこのスクラッチパッドレジスタの間で行われます．なぜ，スクラッチパッドレジスタがこれほどたくさん必要だったかというと，RAMとの間でデータ転送するのがたいへん遅かったためです．RAMの特定の番地からデータを読み書きするには最低でも2命令が必要で20μs以上かかりました（最悪の場合は約100μs）．ですから，たびたび演算の対象になるデータはできるだけCPU内部のレジスタに配置しておけるようにしたのです．

また，スクラッチパッドレジスタは2個ひと組にしてRAMの読み書きをするアドレスを指定するのに使われたり，間接ジャンプのアドレス指定に使ったりできます．単なるポインタなのですが，インテルではインデックスレジスタと呼んでいたようです．

もうひとつ，特殊なレジスタがあります．サブルーチン呼び出しのときに使われるスタックです．4004には4個のアドレススタックがあります．誤解のないように繰り返しますが，スタックポインタでなくて，スタック自身が4個あるのです．サブルーチン呼び出しの戻り番地は，外部のRAM上に記憶させるのではなくて，CPU内部の特殊なレジスタに残します．何といっても，RAMとのデータの入出力は遅いですから，外部メモリに記憶させたのでは効率が上がらなかったでしょう．この4個のアドレススタックのうち，どれかひとつは実行中の命令を指すプログラムカウンタとして用いられていますから，サブルーチン呼び出しは3レベルまでネストできます．プログラムカウンタが4個あって，サブルーチン呼び出しのたびに切り替えられるといったほうが正確かもしれません．

4040のレジスタ構成で変わった点は，インデックスレジスタの数が24個に増えてバンク切り替えが可能になったことと，アドレススタックが8個になってサブルーチン呼び出しが7レベルまで可能になったことです．なにか

図3　4004，4040のレジスタ構成



先ほどのハードウェアのところで説明した割り込みの強化(追加)と裏レジスタの追加，新命令の追加など，まるで8080の改良版として開発されたZ-80とそっくり同じような印象を受けます．

命令セットは4004で46種類，4040で60種類ほどあります．演算命令は4ビットの加減算やローテート命令などが用意されていますが，基本的にアキュムレータだけが対象になっています．スクラッチパッドレジスタに対しては，唯一インクリメントだけが実行できます．また，RAMやI/Oへの入出力は，アドレス指定を行うための命令と，実際に転送を行う命令が別々に存在します．アドレス指定は必ずインデックスレジスタを使う必要があり，直接特定の番地を指定する方法はありません．まず，インデックスレジスタに定数をイミディエートアドレッシングでロードしてから，アドレス指定を行う必要があります．さらにバンク切り替えが必要なら，アキュムレータにバンク切り替え用のデータをロードしてから，バンク切り替え用の命令を実行することになります．なかなかプログラム開発もたいへんだったようですね．

1975年には，テキサス・インスツルメンツ社からTMS1000シリーズという1チップマイクロコンピュータが発表されています．4ビットのマイクロプロセッサは，拡張性が要求されたり大きなプログラムを実行する必要もないのだから，いっそのことメモリも入出力ポートもすべて1つのチップにまとめてしまおうという考え方です．それ以降，この方式が主流となりました．4ビットでCPU部分だけが独立したチップとしては，4040が事実上最後の世代ということになります．結局4ビットのマイクロプロセッサはそれ自身がコンピュータとして表舞台に立つことはなく，ゼンマイ式のタイマーの置き換えなど，深く深く家電の中などへ潜んでいくことになります．

ユーザーがプログラムを書いて実行させられる本来のコンピュータ的な応用が可能になったのは，8ビットマイクロプロセッサが出現してからだといえるでしょう．次回は初期の8ビットマイクロプロセッサの代表，8008と8080あたりを眺めてみましょう．

ICのパッケージはなぜ黒いか

ICやトランジスタのプラスチックパッケージは，ほとんど黒一色で統一されています．黒くないパッケージというとセラミックパッケージか金属のケースに入っているものが一部に使われているだけです．いったいなぜでしょう(ただし，例外というのは必ずあるもので，東芝のPNP型のトランジスタは緑色のプラスチックだし，フォトカプラや，半導体ではないのですが，ICによく似た抵抗アレイは白や黄色になっています)．そういえば，フロッピーディスクもちょっと前までは黒いものばかりでしたが，いまではカラフルなものが売られていますね．半導体製造工場では，コストダウンのために黒に統一しているのでしょうか．種類ごとに色分けすれば一目で見分けがついて便利だと思いませんか．でも，そうしてはいけない理由があるのです．

半導体の中には光に対して反応するものがあります．太陽電池という半導体製品はみなさんもご存知だと思います．光リモコンの受信部などに使われる，フォトトランジスタとかフォトダイオードなど光を検出する特殊なトランジスタやダイオードもあります．また，FAXやビデオカメラの受光部にはCCDという半導体センサが使われています．これらの光に反応する半導体は，一般のトランジスタやダイオード，IC類と原理的にはまったく違いはありません．ふつうのトランジスタのパッケージを無理矢理こじあけて光が入るようにしてやると，光を感じてフォトトランジスタとして働きます．実はコンピュータに使われているIC類も，光に反応してしまうのです．ただ，光検出用の部品は光をたくさん取り入れて効率よく検出する工夫がされているというだけです．

外部の光と無関係な動作をしなければならない一般の半導体類は，光を遮るもので覆ってやらなければいけないのです．そうしなければ，コンピュータルームで記念写真を撮ろうとフラッシュを光らせた瞬間，コンピュータが暴走してしまうかもしれません．だから黒いパッケージなのですね．

デジタルICの中には，データ消去に紫外線を使う種類のROMのようにわざわざガラス窓がついて中が見えるようになっているものもあります．この窓も，ICを動作させるときには必ず遮光シールで覆っておかなければなりません．データが失われる可能性が増すだけでなく，強い光が当たった瞬間に誤動作するおそれがありますので注意してください．実際，その種のROMを直射日光に当ててみたところ，太陽電池と同じように電流を取り出すことができました．

MS-DOS TOOL LABO

MS-DOSツール・ラボ

第5回 スタックとFORTH①

FORTHコンパイラ FAST

加野島英渡

FORTHは，スタックを積極的に利用している風変わりな言語として知られている．しかし，このFORTHを活用することで，強力なアセンブラルーチン管理手法を手に入れることができるのである．今月と来月の2回にわたって，オリジナルFORTHコンパイラFAST(Forth Assembler Sourcecode Translater) の作成を行う．

スタックに光を

スタックは，その重要性から見ると不当なほどに話題にされることが少ない．ほとんど相手にされてこなかったといったほうがいいだろう．通常のプログラミングでスタックを意識するのは，アセンブラならばPUSH／POPを使うときくらいだ．それすらも，データを一時的に退避しておき，必要に応じて復帰するという一時しのぎ的な扱いにすぎない．CやPascalなどの高水準言語にいたっては，スタックを使っていないはずはないのに，その存在すら忘れられている．

忘れられてはいるが，すでに本誌6月号に掲載した再帰的プログラムの項でも説明したように，スタックというのは非常に基本的かつ重要なデータ構造のひとつである．スタックなしにはアセンブラも高級言語も，いやいや，CPUすら語れないというくらいのものだ．家庭でいうと（えっへん）お父さんのような存在なのである．

今月は，ちょっとこの忘れられているお父さん……じゃなかった，スタックというものに光をあててみたい．

スタックとは……

辞書によれば，スタックとは「干し草，麦わらなどの山」なんだそうだ．積み上げられたもの，あるいは積み上げていくことを表現している．

コンピュータのほうでは，データ構造のひとつを表すものとして，このスタックという言葉を使う．この言葉本来の意味どおりに，データを積み上げ，または降ろしていくようなデータ構造をスタックと呼んでいるのだ．積み上げる場所は，メモリである．メモリに

「積み上げる」ということがわかりにくいかもしれないので，ちょっと図1を見ていただこう．

これは，メモリの一部を取り出したものだ．スタックポインタは，このメモリの最新のデータが書き込まれたアドレスを指している．この領域になにかデータを書き込む場合は，まずスタックポインタを決められた方向，たとえばアドレスの若いほうに移動して，そこに書き込む．こうしてスタックポインタをずらしながらどんどんデータを書き込んでいくことができるのだが，この状態がデータを積み重ねていくように見えるということだ．そのつどスタックポインタが移動していくので，図1のように古いデータは失われずにそのまま残っていくことになる．

データを取り出す場合は，まずスタックポインタの指しているデータを取り出し，積み上げたときとは逆方向にスタックポインタを移動する．取り出されたデータはスタックからなくなったものとみなされ，データを取り出す前は古いデータであったものが最新のデータの扱いを受ける（図2）．

このように，スタックでは最後に積まれたデータ（最新のデータ）から取り出されることになる．参照後にスタックポインタが移動してしまうため参照は1回のみ許され，参照されたデータは実際にはメモリに残っているにもかかわらず，スタックポインタの管理から除外されるために，なくなったことになる．

※どうしても参照したあとにそのデータを残したければ，スタック上でいちばん上のデータを複写するという操作をするか，あるいはいったん読んでまた積みなおすということをしなければならない．

こういう構造をLIFO（Last In First Out）あるいはFILO（First In Last Out）と呼んでいる．あとから入ったものが先に取り出され，先に入ったものがあとから取り出されるという，病院の待合室なら喧嘩になりそうな決まりだ．この逆は，FIFO（First In First Out）と呼ぶ．最初に入ったものが最初にとりだされるという，こちらはいわゆる先着順というやつだ．

CPUには，このような機構が内部にハードウェアとして用意されている．使用目的はおもにサブルーチンの呼び出し（CALL）と復帰（RETURN）で，サブルーチンを呼び出すときに戻り番地（呼び出し命令の次のアドレス）をスタックに積み，復帰するときにその戻り番地を取り出す．この場合，CPUに用意されているのはスタックポインタの役割を果たすレジスタだけで，スタック領域は任意のRAM上にとられることが多い．もちろん，8086などのインテル系のCPUでもそうである．

CPUに用意されているこの機構をデータの退避などの目的に使用するための命令が，PUSHとPOPである．CALL／RETURNのように暗黙のうちに使われるにしても，PUSH／POPのように意識して使われるにしても，同じスタックを共用するため，PUSHとPOPあるいはCALLとRETURNはバランスがとれている必要がある．

たとえば，スタックになんらかのデータをPUSHしたままでRETURNしようとすると，スタックのいちばん上のデータを戻り番地と勘違いして，プログラムはいとも簡単に暴走してしまう（はいはい，しょっちゅうやってますよ，すみませんねぇ）．

図1

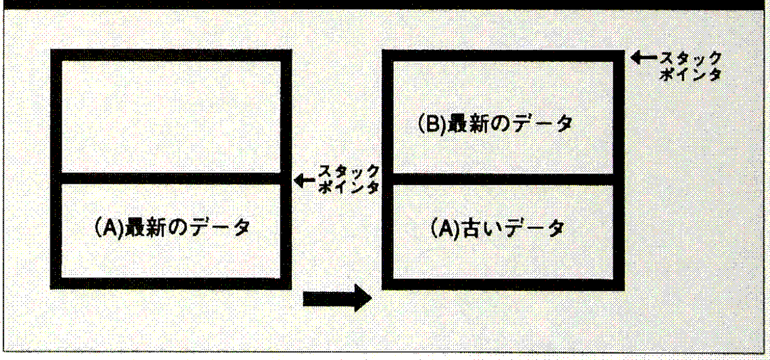
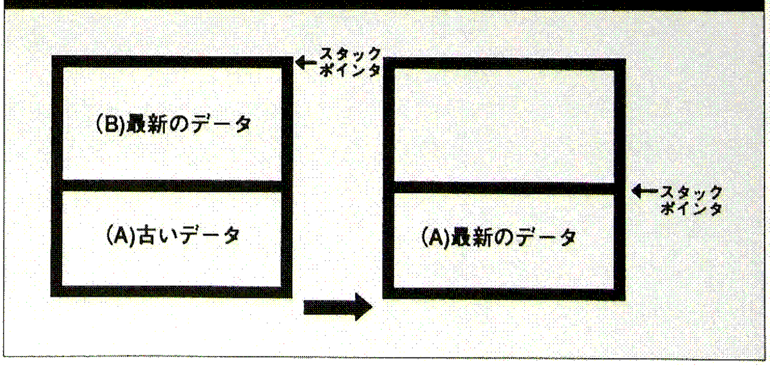


図2



変数とスタックの違い

スタックは，このように記憶領域の使い方のひとつなのだが，一般的な記憶領域の使い方である変数と比べてどのような違いがあるのだろうか.図3にまとめてみたので,ご覧いただきたい.

変数は,「黒板」というイメージである．なにかを書き込むときは前に書いてあるものを消してしまうので，つねに最新の１つのデータしか記憶することができない．これに対してスタックのほうはというと，メモ用紙を何枚も重ねていくという感じになる．メモ用紙の山に新しいメモ用紙を乗せると，今までいちばん上にあったものは見えなくなるが，そのメモ用紙を取り上げると，ついさっきまでいちばん上にあったものがまた顔を見せる．それを取ると，その下のものが最新のデータとして現れるという具合だ.

スタックを語る場合,

- データを発生順にいくつでも記憶していくことができる.
- つねにいちばん新しいものがいちばん取り出しやすい位置にある.

という特性は重要だ.

ところで，プログラムの中で使われるさまざまなデータを寿命（発生してから，消費される，あるいは不要になるまでの期間）という観点から眺めてみよう.

プログラムの実行中にダイナミックに発生するデータというのは，発生と同時に消費されてしまうことが多い．ある演算の結果が次の処理の引数になるというようなものだ．プログラムをローカルに見ていくと，このようなデータの動きが非常に多いことに気がつくだろう．そのプログラムの実行中につねに存在していなければならないデータよりも，このようにできてはすぐに消えていく，泡のようなデータのほうがはるかに多いのである.

もう少し詳しく見ていくと，このようなデータでは消費のされ方がネストしている確率が決して低くないこともわかってくる（図４).

もちろん，すべてがこのようにきれいに動いているわけではないが，このように局所的に発生し，また消費されるデータの存在は否定できない．そして，この記憶場所としてスタックが非常に有効であることも理解できると思う．いちばん消費される確率の高い最新のデータが，いちばん参照しやすいところ，つまりスタックのいちばん上にあるのだから.

ここのところは肝心だから，試験には出ないがしつこく書いておく.

発生したデータの格納場所としてスタックを使うと，最新のデータがいちばん上に置かれることになる．すぐそのあとで消費される場合，データがいちばんアクセスしやすいところにあるので，すぐにとりだすことができる．もうひとつ重要なのは［おいといて］と書いたところで，データが発生してからそれが使われるまでの間に別のデータの発生（と消費）があった場合，それをどこかに退避することなくそのまま［おいといて］をやることができるのである．スタックというひとつの記憶領域をこうした泡のようなデータが共用することができる.

図４

```
データＡの発生　（おいといて）
    データＢの発生
    データＢの消費
    データＣの発生　（おいといて）
        データＤの発生
        データＤの消費
    データＣの消費
データＡの消費
```

もちろんいいことばかりではない．この構造では，古いデータほどアクセスがめんどうになる．机の上ならばドサッと山をひっくり返せば下のデータでもなんとかアクセスすることができるが，スタックの場合は上から順に降ろしていかなければならないので，たいへん効率が悪い．したがって，プログラムの実行中に不定期に，かつひんぱんに参照されるようなデータの格納にはスタックは向いていない．こういう目的に使用する場合には，変数のほうがはるかに有効である.

スタックとFORTH

先に高水準言語ではスタックの存在がほとんど忘れられていると書いたが，ひとつ例外がある．それはFORTHという言語である．伝え聞くところでは，1969年，天文台に勤めるチャールズ・ムーアという暇……あ,

図３

	スタック	変数
参照	1回のみ	何度でも
記憶できるデータ数	無制限	ひとつだけ
データあたりの占有領域	固定	任意

いや，探求心の旺盛な人が望遠鏡の制御プログラムを書くために考えだしたものだそうな．必要は発明の母という言葉はやはりほんとうのようだ．

―― FORTHはスタックを使う ――

「使う」といってもほかの高水準言語のように，暗黙のうちにそのメカニズムを使うというようなものではなく，もっと積極的な意味で使うのである．表舞台の喧騒からはなれて1人舞台廻しをひき受けるスタックを，どのように積極的に使うというのか．

なにやら恐ろしい雰囲気になってきたようだ．一般の高水準言語ではまったく触れていないスタックを正面に据えるFORTHとはどのような言語だろうか．ひとつこの高水準言語のダークサイドを探検してみることにしよう．

FORTHは，CやPascalと同じような手続き型の言語である．ワードと呼ばれる手続きを組み合わせて新しいワードを作成し，それを積み重ねていくことでプログラムを完成する．このあたりはほかの手続き型の言語と事情はまったく同じだが，そのワード間のデータのやり取りにスタックを使うのだ．FORTHではワードへの引数をスタックに積んで渡し，戻り値もまたスタックに積んで返される．その戻り値をまた次に呼び出されるワードが引数として使うという具合になっている．

たとえばCでは，関数呼び出しの際にカッコの中に書かれているものがその関数への引数になる．ずっと前に書かれている関数の結果が，暗黙のうちにどこかの関数に引き継がれるというようなことはない．

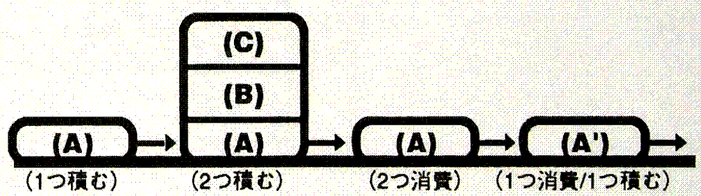
ところがFORTHでは，ワードが呼び出された時点でスタックに積まれているものがそのワードへの引数になる．ここでおもしろいのは，そのデータをスタックに積むのが直前のワードである必要はまったくないということだ．先の［おいといて］である．

次々にワードが実行されていく過程で，スタック（の上のデータ）はダイナミックに生成され，また消費される．あるワードが呼び出されたときに，たまたまスタックに乗っているデータがそのワードへの引数になるのである（図5）．

すでに，発生してすぐに消費されるデータの取り扱いにはスタックがむいていること，そしてそのようなデータの動きはたいへん多いということを説明したが，FORTHはこの点で非常に効率のいい言語であるということができよう．ダイナミックに発生し，消費されるデータを，その格納場所として効率の面でたいへん優れているスタックをそのまま使って受け渡しをするのだから，効率がよくてあたりまえなのである．

……はずなのだが，現在主流になっているCPUはスタックの扱いがあまり効率的ではない．FORTHにとってはたいへん不幸なことである．現在一般に知られているCPUでFORTHに

図5



COLUMN　　FORTH専用CPU

本文で「現在一般的に知られているCPUはFORTHを効率的に動かすようにできていない」と書いたが，それならそういうCPUを作ってしまえとばかりに，チャールズ・ムーア氏が直々にプロジェクトに参画してできたCPUがある．歳のせいでもの忘れがよくなっている（？）ので正式名称は忘れてしまったが，4000番台の番号が付与されたこのCPUは，FORTHを実行するために作られたFORTHのためのCPUである．

いわゆる「FORTHエンジン」であるこのCPUは，リターンスタックから戻り番地を取り出して自分を呼び出したワードに戻るという操作を1命令でやってのける．1命令どころか，CPUコードの中にこの処理専用のビットをもっているのである．

このCPUがFORTHのオブジェクトを実行する能力がどれほどのものかは，フロッピーディスクコントローラというタイミングに厳しいこと鬼のごとくの機能を，このCPUとそのFORTHプログラムで作ることができたという報告が物語っていよう．

昔はこれでもハード屋をめざした筆者は，なんとしてもこれを使ってみたいと思ったものだ．だが，残念ながら日本ではこのCPUは手に入らなかった．いまでもたいへん心残りなことのひとつである．

よく合うものというとZ8000だろうか(ナニ, シラナイ?). このCPUのFORTHコンパイラの生成したオブジェクトを見る機会があったが, たいへんに効率のいいものであった. 68系統のCPUが向いているという話を聞くが, 筆者は68系統にはいままでご縁がなかったので言及する資格がない.

FORTHコンパイラ

FORTHは, コンパイラである. その本体は辞書管理, その実体は仮想スタックマシンである.

最近では, いろいろなタイプのFORTHが出てきているので, 「FORTHとはこれだ」といいにくくなってきているのだが, もっとも基本的なFORTHコンパイラについてみていくことにしよう.

FORTHコンパイラは, ソース中に記述されたワードをコンパイルし, オブジェクトをその名前とともに「辞書」と呼ばれる形式で記憶している. このオブジェクト管理形式がそもそもほかのコンパイラからつまはじきにされそうな構造になっているのだが, これがFORTHのFORTHらしいところなのでしかたがない.

辞書はワードが定義されるにつれて大きく大きく成長していき, 新しいワードはその大きな辞書に定義されているさまざまな単語を引用することで実現される. コンピュータがだんだん賢くなっていくようなもので, まさしくこれは辞書なのである. FORTHのコンパイルとは, 辞書を増殖させていくことにほかならない (図6).

コンパイラはワード名を辞書の中から検索し, それを呼び出すようなオブジェクトを生成する. 辞書の検索は, 最新のワードから開始される. したがって同じ名前で別の定義をすると, それ以降は新しい定義が最初に見つかるので, ワードの意味を変えてしまうことになる. FORTHでは, こんなことすら可能になっているのだ.

オブジェクトはCPUコードそのままではなく, ワードの辞書内での位置を示すデータになっている. これでは, CPUが直接実行するわけにはいかないので, 実行時インタプリタという数ステップのプログラムが, このオブジェクトを実行するようになっている. FORTHは, 実行時インタプリタをCPUと見立てて, そのCPUが実行できるようなオブジェクトを生成するのである.

図7はFORTHのワード定義の典型的な例である. このようにFORTHではワード名を書き, その処理内容を書き連ねていくことで定義を完成させる. もちろん分岐構文も用意されているが, 単に処理 (の名前) を並べていくだけでいいことに注目していただきたい. データは暗黙のうちにスタックが使われるので, 引数というものを書く必要がないのである.

図6のワードは, 図8のようなオブジェクトに展開される.

「なんだ, コンパイラじゃなくてインタプリタじゃないか」といわれるかもしれないが, それは誤解である. コンパイラとインタプリタの違いはソースコードをどの時点で解釈するかで決められるべきもので, オブジェクトの形

図6　FORTHの辞書の構造

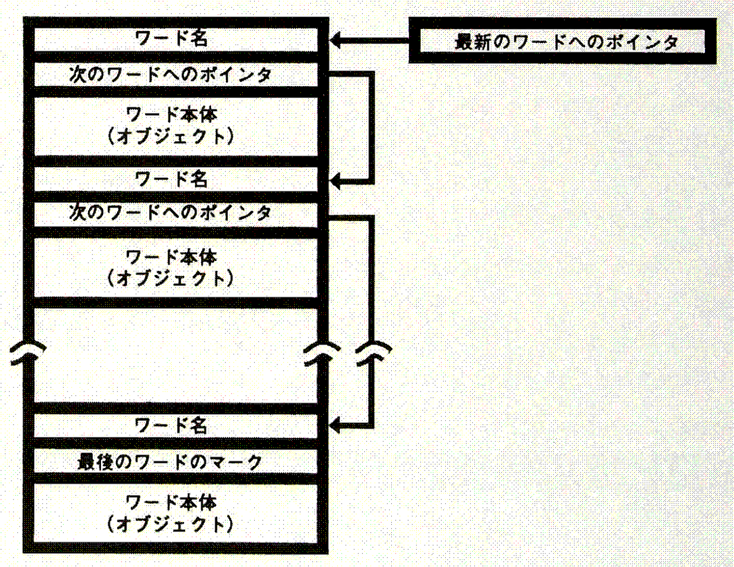
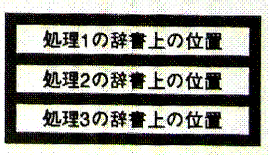


図7

```
[ ワード名
        処理1　処理2　処理3
]
```

図8



*辞書上の位置はアドレス (オフセット) で代表されることが多い。

態や実行形式を分類するものではない。

実行前にすべてのソースコードの解釈を終わらせておくのがコンパイラで，実行時にソースコードの解釈を行うものがインタプリタである．この点でFORTHはまごうことなきコンパイラだ．実行時インタプリタなどというからまぎらわしいのかもしれないが，たとえばこれを「仮想スタックマシン」だの「エグゼキュータ」だのといい換えてみてもやっていることに変わりはない．

この形式のコンパイラを「中間コードコンパイラ」とか「仮想マシンコードコンパイラ」などと呼んでいる．このオブジェクト形式の有名なところでは，Pascalの「Pコード」がある．これに対してCPUが即実行できるようなオブジェクトを出力するものを「ネイティブコードコンパイラ」と呼ぶ．

さて，最近ではこのような古典的な形態のFORTHだけではなく，純粋な意味での辞書（ワードの名前や位置情報など実行時には必要がないもの）とオブジェクトを分離したものや，直接CPUが実行できるようなオブジェクトを生成するものもめずらしくない．しかし，そのような部分がどう変わろうともFORTHはFORTHである．ワード間のデータの受け渡しにスタックを使うという構造であれば，それはFORTHであるといって差し支えない．少なくともFORTH系言語ではある．

ところで，FORTHはワード間のデータの受け渡しにスタックを使うが，このほかにワードの呼び出しと元の位置への復帰のためにもスタックが必要になる．FORTHは，2つのスタックを使うのである．この「2つのスタック」というのが，FORTHのユニークのところなのだ．このそれぞれのスタックを「パラメータスタック」と「リターンスタック」と呼んでいる．FORTH以外の言語やCPUの話では，スタックというとリターンスタックのことを指すが，FORTHでは断りなしにスタックといえばパラメータスタックのことである．

FORTHプログラミング例

参考のためにいくつかのFORTHプログラミング例をお目にかけよう．まずは，2つの数を受けとってその平均値を返すプログラムである（リスト1）．

これだけ？　そう，これだけだ．これを，たとえばCで書くとどうなるだろう（リスト2）．

アセンブラでは，こんな感じになる（リスト3）．

プログラムの動きは，図9のようになる．1つ1つのワード名とスタックの動きを見ていただきたい．スタックの中に書かれているデータは，説明のための仮の数値である．

図9のワードは，スタックに2つの数値が積まれた状態で呼び出される．まず'＋'でこの2つの数値が加算され，結果がスタックに残る．

次に'2'という数値が出てくるが，これは数値の2をスタックに積むというワードである．FORTHでは，数値もそれをスタックに積むという意味のワードとして扱われているのである．

さらに'／'で'＋'の結果を2で割り，その結果がスタックに残る．

スタックに残った結果をそのままにして呼び出し元に戻る．この結果は，このワードを呼び出したワードの次の処理がまた消費することになる．なにに対して処理を行うのか，また，その結果をどこに納めるのかが書かれていないことに注目していただきたい．処理対象はつねにスタックなので，その

リスト1

```
[ average2
        + 2 /
]
```

リスト2

```
int     average2(int a , int b)
{
        return  (a + b) / 2;
}
```

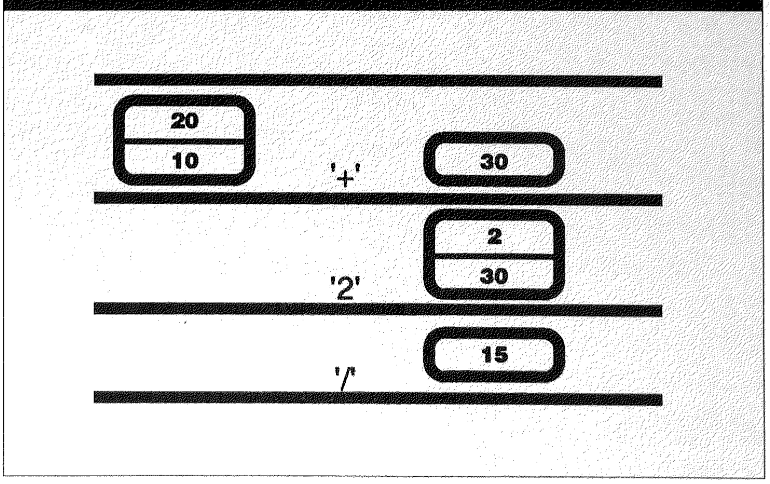
リスト3

```
        add     ax,bx
        shr     ax,1
        ret
```

リスト4

```
[ round
        5 + 10 / 10 *
]
```

図9



ような指定は不要なのである．

もうひとつ見ていただこう，今度は説明は書かないから，どのように動作しているのか考えてみていただきたい（リスト4）．

サブルーチン管理法としてのFORTH

FORTHはコンパイラとしてだけではなく，サブルーチン管理手法として考えることもできる．すべての入力と出力がスタックという共通のインターフェースを通して結びつけられるという構造なので，この構造でどんどんサブルーチンを増やしていけば柔軟で汎用性の高いサブルーチンモジュールが容易に構成できる．アセンブラだと，入力レジスタはこれこれ，出力はALかキャリーかなど，効率を考えるうえでわずらわしい問題にかかわり合わなくてはならない．FORTHでは，この点，すべてスタックを経由する（しなければならない）ので，たいへんすっきりする．

したがって，すべてのサブルーチンをこの構造で作っておけば，単に組み合わせるだけ，順番に呼び出すだけで新しい機能を構成することができる．アセンブラサブルーチンの管理法として，FORTHという言語は強力な環境を提供してくれる．

サブルーチン（ワード）間のインターフェースはスタックを使用すると決められているので，サブルーチンの利用が楽になる．レジスタ渡しだと必要な引数をレジスタにセットするためにレジスタを退避したり移動したりということが必要になるのだが，そういうめんどうがない．しかも，データがスタック上で操作されるために，静的な変数の使用が少なくなり，サブルーチンの独立性が高まり，再利用がしやすくなる．

ワードの呼び出しは，実行時インタプリタ方式でもほんの数バイトのオーバヘッドですむし，もしそれがいやなら，直接CPUのCALL命令を生成するようにしてやれば，速度の面でも遜色ないものができる．もちろん，しっかりとアセンブラで書いたものには遠くおよばないが，そのぶん楽ができるのだから，それを速度に影響する部分にかけてやればいいのである．

FORTH系言語【FAST】を作る

なかなかおもしろそうなので，FORTHコンパイラを書いてみようと思う．といっても，この連載はアセンブラ講座なのだから，いくらアセンブラのサブルーチンをきれいに管理できるからといって，単なるFORTHコンパイラを書いたのでは読者に申しわけが立たない．なにか工夫をしてみよう．

歳がばれるのを恐れずにいえば，筆者とFORTH系言語とのつき合いはもうかれこれ10年近くになろうとしている．なんの基礎もできていない筆者にアセンブラというものを最初に教えてくださった方が，雑誌に載っていたZ80のアセンブラで書かれた簡単なFORTHコンパイラのソースを教材に取りあげられて以来，このユニークで魅力的な言語とのつき合いは，絶えることなく今日まで続いている．

この間に2本のFORTHコンパイラを書き，実際に仕事でも使っていた立場から，この言語には魅力と同じくらい不満もある．

まず第一に前方参照ができない．参照されるラベルは必ず参照前に定義されている必要がある．辞書に定義されているワードを使って上位のワードを定義するのだから，これは当然だ．早い話が，最初に実行されるワードはソースのいちばん最後に書かなければならないのだ．

たいていの場合は，これでも不自由を感じることはない．筆者など，FORTHでのプログラミング経験が長かったために，プログラムを書く場合にはまず使用する部品から書いていく癖が身についてしまっている．しかしそうではない人も多いのだし，ワードを置く場所が制限を受けるというのはあまりおもしろいことではない．

次に，アセンブラとのリンクがやりにくい．たいていのFORTHコンパイラはアセンブラ機能を内蔵しているが，MASMで書かれたプログラムとリンクしようとすると，そのFORTHコンパイラの記法にしたがってソースを書きなおさなくてはならない．ましてやマイクロソフトのリンカなど使えないので不自由このうえない．

リンカが使えないくらいだから，当然ライブラリマネージャも使えない．必要なワードだけを選択的にリンクするという結構な手法が使えないというのは痛い．FORTHの場合は前方参照ができないという決まりなので，あるワードが参照されるときにはそのワードはすでに定義ずみであることがあたりまえだから，とりあえず参照を棚上げにしておいてあとから解決しようという発想が出てこなかったのだろう．あとまわしの発想がないのならば，ライブラリから選択的にリンクするという手法が用意されていなかったというのも別段不思議なことではない．

しかし，不思議ではないが不便ではある．いくつかのワードをコンパイルして辞書を構成し，その辞書を使って新しいワードをコンパイルしていくという構造なので，あってはならないことだが，下のほうのワードにバグでも

あった日にはそのレベルからコンパイルのやりなおしである．

このあたりをなんとか解決できないものだろうか？　それも「簡単に」だ．手間をかければたいていのことはできるのだろうが，ちょっとうまい手を考えて，簡単にすませるというのがカッコいいのだ．

オブジェクトとして，アセンブラのソースを吐き出すというのはどうだろう．コンバータというかトランスレータというか……，FORTH形式の文法をもったアセンブラソースコードジェネレータだ．FORTH形式でソースを書くと，MASMでアセンブル可能なソースコードを吐き出してくれる．これならば前方参照は思いのままだし(コンバータがチェックをサボればという話だが)，すでにMASMで書かれているプログラムとのリンクも，データの受け渡しさえ合わせてやればマイクロソフトのリンカを使うことができるし，ライブラリマネージャの使用も可能だ．実際のオブジェクトコードの生成とエラーチェックはアセンブラがやってくれるので，そのぶんコンパイラが小さくてすむことにもなる．いやいや，まったく手抜きにかけては天才的な才能があるなぁ．

FASTの設計

まずは，名前．このコンパイラを「FAST」と名付けよう．“Forth Assembler Sourcecode Translater”というこじつけだ．

FASTは，FORTH系言語の特色である小回りのきく記述が可能なように，できるだけ自由度の大きい設計にしたいと思う．それを実現するには，よけいな手出しをしないことだ．

文法をがっちり決めてしまって，すべてをFASTの世界にもち込むのではなく，アセンブラのソースの中に必要に応じてFORTH記述を入れることができるという構造にしておこう．FAST自身は，FORTH記述の部分をアセンブラのソースに置き換えるというフィルタ的な作業に専念し，それ以外の部分には介入しないことにする．

こうすることによって，なれないうちはすべてをアセンブラ記述ですませ，なれるにしたがって徐々にFORTH記述を増やしていくというスムーズな移行が可能になる．しかも，アセンブラ記述はMASMの文法そのものである(FORTH記述以外の部分は素通りにするのだから，あたりまえといえばあたりまえだ)．

■2つのモード

FASTは，アセンブラモードとFORTHモードの2つのモードをもつ．起動時はアセンブラモードで，FORTHモードに移行するトリガを発見するとFORTHモードに切り替わる．FORTHモードの中で，アセンブラモードに移行するトリガを発見すると元のアセンブラモードに戻るようにする．

FORTHモードの中では，すべてのワード（アセンブラに移行するトリガを除く）はFORTHのワードとして認識される．

■ワードの認識

TABまたは空白をワードの区切りとする．ただし，‘(’かシングルクォート，ダブルクォートで囲まれたものは間に空白を含んでいてもよい．

■ワードの属性

ダブルクォートで囲まれたものは，ASCIZ文字列（0で終端された不定長の文字列）である．すべてが数字であるもの，または‘(’かシングルクォートで囲まれた単語は数値と解釈する．#ではじまるものはアドレス（オフセット値)，それ以外はワードの呼び出しと解釈する．

FASTは，FORTH記述以外の部分には関与しないので，変数(DB，DWなどで記述したもの）名を書いてもそれが変数であるという認識ができない．そのままではその変数名を呼び出すようなソースを出してしまうので，これを‘#’によって区別させる．

■ラベルの扱い

FORTHでは，‘+’や‘.’など，MASMが使用を禁止しているものも名前として使うことができる．これらの記号を名前の中に許すことで，より直感的な名前をつけることが可能になっている．これは採用したい．

そこで，MASMがラベルとして許していないキャラクタは，なんらかの法則で別の文字あるいは文字列に変換してしまうことにする．生成されたオブジェクト（つまりMASMのソース）はふだんは見ないのだから，多少わかりにくくなっても大丈夫だ．

また，FORTH標準のワード名の中には，アセンブラが識別子として使っている文字列とぶつかり合うものもある．“DUP”，“LOOP”などだが，これは，FORTHモードではすべての名前の前に強制的になんらかの文字列を付加することで，ぶつかりあいを避けることにする．

レジスタ/バッファ割り当て

DS，CS，SSは，ライブラリ内では保存される．SPはリターンスタックポインタ，BPは0からSPとは逆方向に

使われるパラメータスタックポインタである（図10）．

パラメータスタックのデータサイズは，ダブルワード（32ビット）とする．ただし，演算や比較はワード（16ビット）単位の無符号で行う．パラメータスタックの上位ワードには通常はセグメント値が入るが，メモリアクセス以外では無視される．

■ワード呼び出しのオブジェクト生成

'#'がなければ，変数名にも同じオブジェクトを生成するので注意のこと（図11）．

■数値のオブジェクト生成

数値とは，0～9ではじまるものか，'('またはシングルクォートで囲まれた

図10　スタックの配置

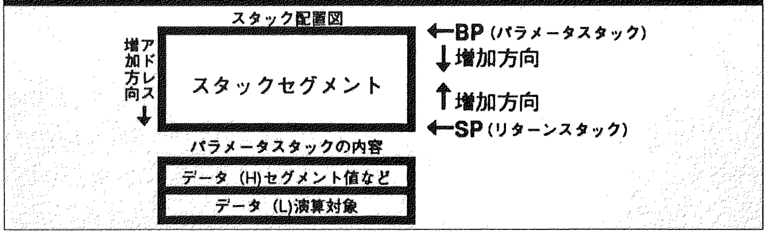


図11　ワード呼び出し

"CALL <名前>"

図12　数値のオブジェクト生成

"CALL　　$$WNUMBER"
"DW　　<数字>" または "DW　　<名前>"

リスト5　FASTによるメモリダンププログラム

```
; ============================================================
;
; ＦＡＳＴサンプルプログラム "DUMP.F"
;
;        Copyright(C) 1991 By Justech Labo. H.Kanoshima
;
; ============================================================

include         core.h

_TEXT   SEGMENT WORD PUBLIC 'CODE'
        ASSUME  CS:_TEXT , DS:_DATA

TRUE    equu    1
FALSE   equ     0

[ read_dumpdata         ; . | *(dump_off++)
        #dump_off @1+ #dump_seg @ full_address c@
]

[ nhex_s+               ; data , address | .
        swap
        '0' + dup '9' >
        if      7 +
        then
        swap
        c+
]

[ bhex_s+               ; data , address | .
        over over
        swap 4 >> swap          nhex_s+
        swap 0fh & swap         nhex_s+
]

[ whex_s+               ; data , address | .
        over over
        swap 8 >> swap          bhex_s+
        swap 0ffh & swap        bhex_s+
]

[ check_char            ; char | char (convert unprintable char to '.')
        7fh & dup ' ' <
        if      drop '.'
        then
]

[ dump_hex      ; . | .
        #dump_off @
        16 do
                read_dumpdata #print_buf bhex_s+
                ' ' #print_buf c+
        loop
        #dump_off !
]

[ dump_ascii    ; . | .
        #dump_off @
        16 do
                read_dumpdata check_char #print_buf c+
        loop
        #dump_off !
]

[ dump_line             ; . | .
        0 #print_buf c!
        #dump_seg @ #print_buf whex_s+
        ':' #print_buf c+
        #dump_off @ #print_buf whex_s+
        ' ' #print_buf c+
;
        dump_hex dump_ascii
        16 #dump_off +!
;
        ' ' #print_buf c+
        #print_buf sprint crlf
]

[ ?hexnumber    ; ascii-code | TRUE/FALSE
        cupcase
        dup 'A' 'F' <range>
        if      drop (TRUE)
        else    '0' '9' <range>
                if      (TRUE)
                else    (FALSE)
                then
        then
]

[ hexnumber     ; ascii-code | hexnumber
        cupcase
        '0' - dup 9 >
        if      7 -
        then
]

[ get_hex       ; address | next-address data
        0 >r
        begin
                dup c@ dup ?hexnumber
                if      hexnumber r> 4 << + >r
                        1+      (TRUE)
                else    drop    (FALSE)
                then
        until
        r>
]

[ skip_space    ; address | address
        begin
                dup c@ ' ' =
                if      1+      (TRUE)
                else            (FALSE)
                then
        until
]

[ main          ; This is main
        _startup
        my_psp 81h + skip_space get_hex #dump_off !
        dup c@ ':' =
        if      #dump_off @ #dump_seg !
                1+ get_hex #dump_off !
        then
        drop
        16 do
                dump_line
        loop
]

_TEXT   ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'

dump_off        dw      (?)
dump_seg        dw      (?)

print_buf       db      80 dup (?)

inpbuf          db      256 dup (?)

_DATA   ENDS

        END     f_main

```


ワンラインNEWS
MCUは現在，4ビットから32ビットまで市場に出ているが，好調なのは相変わらず4ビットである．家電などを中心に高い伸びを見せている．

もの．#つきの変数名も同じ．スタックの上位ワード（スタックは32ビットである）には自動的にDSの値が積まれる．'#'がついたワードに対しては，図13のようなオブジェクトを生成する．“$$WNUMBER”は自分の直後のワード値をスタックに積むワードである．

■文字列のオブジェクト生成

文字列がヌルの場合は，“～”は生成されない．“$$CSTRING”は，自分の直後のASCIZ文字列をスタックに積むワードである（図14）．スタックの上位ワード（スタックは32ビットである）には，このワードの中で自動的にCSの値が積まれる．

図13　'#'のついたワード

```
"CALL      $$WNUMBER"
"DW        OFFSET  <名前>"
```

図14　文字列のオブジェクト生成

```
"CALL      $$CSTRING
"DB        "～",0
```

■分岐ワードのオブジェクト生成

スタックトップが真（0以外の数値）ならば，THENまでの間のワードを実行する．さもなければ，THENの次のワードから実行する．“$$JMPFALSE”は，スタックトップが0ならば直後のワード値をオフセットとしてそこに制御を移すワードである（図15）．

スタックトップが真（0以外の数値）ならばELSEまでの間のワードを実行し，THENの次のワードへ，さもなければELSEの次のワードから実行する（図16）．

UNTILにきた時点でスタックトップが真ならばBEGINに戻る．

図15　分岐ワード　IF～THEN

```
    "CALL      $$JMPFALSE"        ; IF
    "DW        $0000

         ・・・・

"$0000"                           ; THEN
```

図16　分岐ワード　IF～ELSE～THEN

```
    "CALL      $$JMPFALSE"        ; IF
    "DW        $0000

         ・・・・

    "JMP       $0001              ; ELSE
"$0000:"

         ・・・・

"$0001"                           ; THEN
```

図17　分岐ワード　BEGIN～UNTIL

```
"$0000:"                                    ; BEGIN

      ・・・・

      "CALL      $$JMPTRUE          ; UNTIL
      "DW        $0000"
```

リスト6　FASTによる出力ファイル

```
; =============================================================
;
; ＦＡＳＴサンプルプログラム "DUMP.F"
;
;        Copyright(C) 1991 By Justech Labo. H.Kanoshima
;
; =============================================================

include         core.h

_TEXT   SEGMENT WORD PUBLIC 'CODE'
        ASSUME  CS:_TEXT , DS:_DATA

TRUE    equu    1
FALSE   equ     0

f_read_dumpdata proc
if      0
extrn   f_read_dumpdata:near
endif
public  f_read_dumpdata
;dump.f(18)
;       #dump_off @1+ #dump_seg @ full_address c@
        call    f_$$wnumber
        dw      offset dump_off
        call    f__40_1_2B_
        call    f_$$wnumber
        dw      offset dump_seg
        call    f__40_
        call    f_full_address
        call    f_c_40_
;dump.f(19)
;]
        ret
f_read_dumpdata endp


f_nhex_s_2B_    proc
if      0
extrn   f_nhex_s_2B_:near
endif
public  f_nhex_s_2B_
;dump.f(22)
;       swap
        call    f_swap
;dump.f(23)
;       '0' + dup '9' >
        call    f_$$wnumber
        dw      '0'
        call    f__2B_
        call    f_dup
        call    f_$$wnumber
        dw      '9'
        call    f__3E_
;dump.f(24)
;       if      7 +
        call    f_$$jmpfalse
        dw      $0000
        call    f_$$wnumber
        dw      7
        call    f__2B_
;dump.f(25)
;       then
$0000:
;dump.f(26)
;       swap
        call    f_swap
;dump.f(27)
;       c+
        call    f_c_2B_
;dump.f(28)
;]
        ret
f_nhex_s_2B_    endp


f_bhex_s_2B_    proc
if      0
extrn   f_bhex_s_2B_:near
endif
public  f_bhex_s_2B_
;dump.f(31)
;       over over
        call    f_over
        call    f_over
;dump.f(32)
;       swap 4 >> swap                  nhex_s+
        call    f_swap
        call    f_$$wnumber
        dw      4
        call    f__3E__3E_
        call    f_swap
        call    f_nhex_s_2B_
;dump.f(33)
;       swap 0fh & swap                 nhex_s+
        call    f_swap
        call    f_$$wnumber
        dw      0fh
        call    f__26_
        call    f_swap
        call    f_nhex_s_2B_
;dump.f(34)
;]
        ret
f_bhex_s_2B_    endp


f_whex_s_2B_    proc
if      0
extrn   f_whex_s_2B_:near
endif
public  f_whex_s_2B_
;dump.f(37)
;       over over
        call    f_over
        call    f_over
;dump.f(38)
;       swap 8 >> swap                  bhex_s+
        call    f_swap
        call    f_$$wnumber
        dw      8
        call    f__3E__3E_
        call    f_swap
        call    f_bhex_s_2B_
;dump.f(39)
;       swap 0ffh & swap                bhex_s+
        call    f_swap
        call    f_$$wnumber
        dw      0ffh
        call    f__26_
        call    f_swap
        call    f_bhex_s_2B_
;dump.f(40)
;]
        ret
f_whex_s_2B_    endp


f_check_char    proc
if      0
extrn   f_check_char:near
endif
public  f_check_char
;dump.f(43)
;       7fh & dup ' ' <
        call    f_$$wnumber
        dw      7fh
        call    f__26_
        call    f_dup
        call    f_$$wnumber
```

ワンラインNEWS
著作権使用許可の委託を受けている日本音楽著作権協会は著作権使用の申請を，従来の書類のみからフロッピーディスクや磁気テープでも受け付けるようになった．

“$$JMPTRUE”は，スタックトップが0以外ならば直後のワード値をオフセットとしてそこに制御を移すワードである（図17）．

スタックトップの数だけ“DO”～“LOOP”の間を実行する．“$$DO_ENTRY”は，ループカウンタの初期値（0固定）とループの回数（スタックトップの数値）をリターンスタックに積むワード．“$$LOOP_ENTRY”は，リターンスタック上のループカウンタを増加し，ループ回数と比較して回数に満たない場合は直後のワード値をオフセットとしてそこに制御を移し，回数分繰り返したらリターンスタック上の2つのデータを捨てるワードである（図18）．

図18 分岐ワードDO～LOOP

```
    "CALL      $$DO_ENTRY"      ; DO
"$0000:"
              ・・・・
    "CALL      $$LOOP_ENTRY     ; LOOP
    "DW        $0000"
```

■そのほか

‘;’以降は，コメントとして扱われる．また，先頭が‘*’である行は，‘*’を除いてそのまま出力される．これにより，FORTH記述部分の中に随時アセンブラ記述文を挿入することができる．コア（基本ライブラリ）の大部分は，この機能によってアセンブラで記述されることになる．

サンプルプログラムと生成コード

FASTでメモリダンププログラムを書いてみたので，ご覧いただこう．コマンドラインオプションでアドレスを指定することができる．指定しなければ，0：0（割り込みベクタ付近）を表示する．表示範囲は256バイトだが，これは簡単にすませたためで，範囲を指定できるように改造することはもちろん可能だ．

FASTがこのソースをどのようにコンパイルするかを見ていただくために，FASTによって出力されたファイルもいっしょに掲載する．ご覧のように各ワードは単純にCALL命令に展開される．FORTH記述部分のソースはコメントの形で出力されているので，分岐構文のコンパイルも，どのようになっているのかよくわかると思う．

ただ，残念だがこれをこのままアセンブルしてもコア（基本ライブラリ）がないのでリンクができない．したがって実行形式のオブジェクトが作れない．まことにもうしわけないが，雰囲気を味わうところで我慢していただきたい．来月は，このFAST自身で記述したFASTコンパイラと，コアと呼ばれる基本ライブラリを一挙掲載する予

```
;       dw      ' '
        call    f__3C_
;dump.f(44)
;       if      drop ' '
        call    f_$$jmpfalse
        dw      $0001
        call    f_drop
        call    f_$$wnumber
        dw      ' '
;dump.f(45)
;       then
$0001:
;dump.f(46)
;]
        ret
f_check_char    endp


f_dump_hex      proc
if      0
extrn   f_dump_hex:near
endif
public  f_dump_hex
;dump.f(49)
;       #dump_off @
        call    f_$$wnumber
        dw      offset dump_off
        call    f__40_
;dump.f(50)
;       16 do
        call    f_$$wnumber
        dw      16
        call    f_$$do_entry
$2002:
;dump.f(51)
;               read_dumpdata #print_buf bhex_s+
        call    f_read_dumpdata
        call    f_$$wnumber
        dw      offset print_buf
        call    f_bhex_s_2B_
;dump.f(52)
;               ' ' #print_buf c+
        call    f_$$wnumber
        dw      ' '
        call    f_$$wnumber
        dw      offset print_buf
        call    f_c_2B_
;dump.f(53)
;       loop
        call    f_$$loop_entry
        dw      $2002
;dump.f(54)
;       #dump_off !
        call    f_$$wnumber
        dw      offset dump_off
        call    f__21_
;dump.f(55)
;]
        ret
f_dump_hex      endp


f_dump_ascii    proc
if      0
extrn   f_dump_ascii:near
endif
public  f_dump_ascii
;dump.f(58)
;       #dump_off @
        call    f_$$wnumber
        dw      offset dump_off
        call    f__40_
;dump.f(59)
;       16 do
        call    f_$$wnumber
        dw      16
        call    f_$$do_entry
$2003:
;dump.f(60)
;       read_dumpdata check_char #print_buf c+
        call    f_read_dumpdata
        call    f_check_char
        call    f_$$wnumber
        dw      offset print_buf
        call    f_c_2B_
;dump.f(61)
;       loop
        call    f_$$loop_entry
        dw      $2003
;dump.f(62)
;       #dump_off !
        call    f_$$wnumber
        dw      offset dump_off
        call    f__21_
;dump.f(63)
;]
        ret
f_dump_ascii    endp


f_dump_line     proc
if      0
extrn   f_dump_line:near
endif
public  f_dump_line
;dump.f(66)
;       0 #print_buf c!
        call    f_$$wnumber
        dw      0
        call    f_$$wnumber
        dw      offset print_buf
        call    f_c_21
;dump.f(67)
;       #dump_seg @ #print_buf whex_s+
        call    f_$$wnumber
        dw      offset dump_seg
        call    f__40
        call    f_$$wnumber
        dw      offset print_buf
        call    f_whex_s_2B_
;dump.f(68)
;       ':' #print_buf c+
        call    f_$$wnumber
        dw      ':'
        call    f_$$wnumber
        dw      offset print_buf
        call    f_c_2B_
;dump.f(69)
;       #dump_off @ #print_buf whex_s+
        call    f_$$wnumber
        dw      offset dump_off
        call    f__40
        call    f_$$wnumber
        dw      offset print_buf
        call    f_whex_s_2B_
;dump.f(70)
;       ' ' #print_buf c+
        call    f_$$wnumber
        dw      ' '
        call    f_$$wnumber
        dw      offset print_buf
        call    f_c_2B_
;dump.f(71)
;;
;dump.f(72)
;       dump_hex dump_ascii
        call    f_dump_hex
        call    f_dump_ascii
;dump.f(73)
;       16 #dump_off +!
        call    f_$$wnumber
        dw      16
        call    f_$$wnumber
        dw      offset dump_off
        call    f__2B__21_
;dump.f(74)
;;
;dump.f(75)
;       ' ' #print_buf c+
        call    f_$$wnumber
        dw      ' '
        call    f_$$wnumber
```

定である。

今月の反省

アセンブラの話だと思っていたらいきなりスタックで，あげくの果てにはFORTHコンパイラだ．さぞかし驚かれた読者も多いことだろうが,「この命令をこう使うとこんな手品ができます」なんて話は，それはそれでたいそうおもしろいのだが，興味のあるひとが考えてくれればいいことだ．アセンブラという言語を通して,コンピュータのいろいろなおもしろさを少しでも味わっていただきたいと思うのである．

```
        dw      offset print_buf
        call    f_c_2B_
;dump.f(76)
;       #print_buf sprint crlf
        call    f_$$wnumber
        dw      offset print_buf
        call    f_sprint
        call    f_crlf
;dump.f(77)
;]
        ret
f_dump_line     endp
f_?hexnumber    proc
if      0
extrn   f_?hexnumber:near
endif
public  f_?hexnumber
;dump.f(80)
;       cupcase
        call    f_cupcase
;dump.f(81)
;       dup 'A' 'F' <range>
        call    f_dup
        call    f_$$wnumber
        dw      'A'
        call    f_$$wnumber
        dw      'F'
        call    f__3C_range_3E_
;dump.f(82)
;       if      drop (TRUE)
        call    f_$$jmpfalse
        dw      $0004
        call    f_drop
        call    f_$$wnumber
        dw      (TRUE)
;dump.f(83)
;       else    '0' '9' <range>
        jmp     $0005
$0004:
        call    f_$$wnumber
        dw      '0'
        call    f_$$wnumber
        dw      '9'
        call    f__3C_range_3E_
;dump.f(84)
;               if      (TRUE)
        call    f_$$jmpfalse
        dw      $0006
        call    f_$$wnumber
        dw      (TRUE)
;dump.f(85)
;               else    (FALSE)
        jmp     $0007
$0006:
        call    f_$$wnumber
        dw      (FALSE)
;dump.f(86)
;               then
$0007:
;dump.f(87)
;       then
$0005:
;dump.f(88)
;]
        ret
f_?hexnumber    endp
f_hexnumber     proc
if      0
extrn   f_hexnumber:near
endif
public  f_hexnumber
;dump.f(91)
;       cupcase
        call    f_cupcase
;dump.f(92)
;       '0' - dup 9 >
        call    f_$$wnumber
        dw      '0'
        call    f__2D_
        call    f_dup
        call    f_$$wnumber
        dw      9
        call    f__3E_
;dump.f(93)
;       if      7 -
        call    f_$$jmpfalse
        dw      $0008
        call    f_$$wnumber
        dw      7
        call    f__2D_
;dump.f(94)
;       then
$0008:
;dump.f(95)
;]
        ret
f_hexnumber     endp
f_get_hex       proc
if      0
extrn   f_get_hex:near
endif
public  f_get_hex
;dump.f(98)
;       0 >r
        call    f_$$wnumber
        dw      0
        call    f__3E_r
;dump.f(99)
;       begin
$1009:
;dump.f(100)
;               dup c@ dup ?hexnumber
        call    f_dup
        call    f_c_40_
        call    f_dup
        call    f_?hexnumber
;dump.f(101)
;               if      hexnumber r> 4 << + >r
        call    f_$$jmpfalse
        dw      $000A
        call    f_hexnumber
        call    f_r_3E_
        call    f_$$wnumber
        dw      4
        call    f__3C__3C_
        call    f__2B_
        call    f__3E_r
;dump.f(102)
;                       1+      (TRUE)
        call    f_1_2B_
        call    f_$$wnumber
        dw      (TRUE)
;dump.f(103)
;               else    drop    (FALSE)
        jmp     $000B
$000A:
        call    f_drop
        call    f_$$wnumber
        dw      (FALSE)
;dump.f(104)
;               then
$000B:
;dump.f(105)
;       until
        call    f_$$jmptrue
        dw      $1009
;dump.f(106)
;       r>
        call    f_r_3E_
;dump.f(107)
;]
        ret
f_get_hex       endp
f_skip_space    proc
if      0
extrn   f_skip_space:near
endif
public  f_skip_space
;dump.f(110)
;       begin
$100C:
;dump.f(111)
;               dup c@ ' ' =
        call    f_dup
        call    f_c_40_
        call    f_$$wnumber
        dw      ' '
        call    f__3D_
;dump.f(112)
;               if      1+      (TRUE)
        call    f_$$jmpfalse
        dw      $000D
        call    f_1_2B_
        call    f_$$wnumber
        dw      (TRUE)
;dump.f(113)
;               else            (FALSE)
        jmp     $000E
$000D:
        call    f_$$wnumber
        dw      (FALSE)
;dump.f(114)
;               then
$000E:
;dump.f(115)
;       until
        call    f_$$jmptrue
        dw      $100C
;dump.f(116)
;]
        ret
f_skip_space    endp
f_main  proc
if      0
extrn   f_main:near
endif
public  f_main
;dump.f(119)
;       _startup
        call    f__startup
;dump.f(120)
;       my_psp 81h + skip_space get_hex #dump_off !
        call    f_my_psp
        call    f_$$wnumber
        dw      81h
        call    f__2B_
        call    f_skip_space
        call    f_get_hex
        call    f_$$wnumber
        dw      offset dump_off
        call    f__21_
;dump.f(121)
;       dup c@ ':' =
        call    f_dup
        call    f_c_40_
        call    f_$$wnumber
        dw      ':'
        call    f__3D_
;dump.f(122)
;       if      #dump_off @ #dump_seg !
        call    f_$$jmpfalse
        dw      $000F
        call    f_$$wnumber
        dw      offset dump_off
        call    f__40_
        call    f_$$wnumber
        dw      offset dump_seg
        call    f__21_
;dump.f(123)
;               1+ get_hex #dump_off !
        call    f_1_2B_
        call    f_get_hex
        call    f_$$wnumber
        dw      offset dump_off
        call    f__21_
;dump.f(124)
;       then
$000F:
;dump.f(125)
;       drop
        call    f_drop
;dump.f(126)
;       16 do
        call    f_$$wnumber
        dw      16
        call    f_$$do_entry
$2010:
;dump.f(127)
;               dump_line
        call    f_dump_line
;dump.f(128)
;       loop
        call    f_$$loop_entry
        dw      $2010
;dump.f(129)
;]
        ret
f_main  endp
_TEXT   ENDS
_DATA   SEGMENT WORD PUBLIC 'DATA'
dump_off        dw      (?)
dump_seg        dw      (?)
print_buf       db      80 dup (?)
inpbuf          db      256 dup (?)
_DATA   ENDS
        END     f_main
```

98活用アセンブラ技法

ビデオシステム編
第8回

バックカラー表示
常駐プログラムの作成

星野 操

前回までに，グラフィック画面を利用してバックカラーつきで文字表示を行うルーチンを示しました．今回は，DOSの文字表示に使用されるINT 29Hをトラップして，DOSレベルでバックカラー表示を可能にする常駐プログラムを作成します．

前回までのプログラムの修正

バグの修正とプログラムの追加

resident.asm

今回使用するモジュールで，前回以前に掲載されたものを修正しておきます．

まず，resident.asmについて，46行目に，リスト1の1行を追加します．これは，今回のプログラムで割り込みのチェーンを実行するために使用するものです．

次に，プログラム中で3か所ほど記載のあるリスト2のような記述をリスト3のように変更します．Turbo Assemblerでは，リスト2のような記述が許されますが，Macro Assemblerではbxが [] のなかに存在していないとエラーになるようです．Turbo Assemblerを使用する場合はとくに修正する必要はありません．

最後に400行目にあるリスト4の直前にリスト5を追加，402行目のリスト6の直後にリスト7を追加します．せっかく計算した常駐パラグラフサイズがベクタの復元によって破壊されてしまっています．これは単純な誤りでしたが，前回はDXに設定されている値より必要な常駐サイズが少ないので見落としていました．

back2.asm

次に，back2.asmの修正ですが，前回の修正に加えて354行付近のリスト8の直後にリスト9を加えてください．

リスト1
```
public  my_vect
```
リスト2
```
strncmp <offset r_option,bx[2],2>        ; compare option
```
リスト3
```
strncmp <offset r_option,[bx+2],2>        ; compare option
```
リスト4
```
lds     dx,vect23
```
リスト5
```
push    dx
```
リスト6
```
dos     <25h,,,>
```
リスト7
```
pop     dx
```

プログラムの仕様

常駐プログラムでバックカラーを表示

さて，前回作成したウィンドウライブラリでは，これを利用するプログラムからしかバックカラーを使用することはできませんでした．これではかなり使用が限定されてしまいます．そこで，今回はもう少し一般的に使用できるように，常駐プログラムで通常のDOS出力にバックカラーをつけられるようにしてみました．

PC-9801用の現在のDOSでは，コンソールへの出力は通常INT 29Hを経由して実行されます．ただし，これが満たされない環境も存在します．この割り込みは，画面出力を高速に行うためのもので，通信プログラムなどでは画面表示にDOSの文字出力を使用せず直接INT 29Hを発行するものもあります（もちろん，さらに高速化を目指す場合には画面に直接書き込みをすることになります）．

INT DCHで文字表示を行わせることもできますが，これはそれほど重視する必要はないでしょう．けっして完全ではありませんが，INT 29Hをトラップして自前でバックカラーつきの文字表示を行うことで，バッチファイルのメッセージ出力や簡単なDOSコールのみを使用しているプログラムでもエスケープシーケンスでバックカラーを表示できるようになります．

IBM PCの標準コンソールドライバはエスケープシーケンスをサポートしていませんが，配布されている拡張コンソールドライバでは，バックカラーとフォアカラーを独立して指定できるはずです．

PC-9801では，esc+[とmの間の数

リスト8

```
neg     col
```

リスト9

```
inc     col
```

値が30～37で指定色の通常表示，40～47で指定色のリバース表示を行うようになっています．ハードウェアの制限から，これらを独立に指定できないので，一方を指定すれば他方が意味をもたなくなります．バックカラーをサポートするためには，これらのシーケンスを別個に解釈して前者をフォアカラー，後者をバックカラーとして表示すればよいことになります．

ただし，これらのエスケープシーケンスを解釈するためには，そのほかのエスケープシーケンスもきちんと解釈しないといけません．バックカラーの表示だけ自前で担当し，文字の表示やそのほかのエスケープシーケンスのコントロールなどはもともとのINT 29Hドライバに任せるということも考えられます．しかし，たとえば，PC-9801では漢字の中央で折り返しが生じる場合には半角スペースを送ってそのような現象が生じないように補正する，というようにいろいろな表示のステートを記憶／処理しないと元のドライバとのマッチングが取れなくなってしまいます．

このため，現実問題としてはINT 29Hを完全に乗っ取り，文字表示に関することはすべて自前で行う必要があるのです．解釈すべきエスケープシーケンスは多岐におよびますが，ここでは参考文献(1)に記述されている範囲のエスケープシーケンスを基本的にサポートすることにします．

テーブルの使用

今回のプログラムでは，今まで使用しなかったテーブルを使用してみようと思います．以前述べたようにテーブルは，プログラムの高速化の手法ですが，その分メモリサイズを大きくしてしまう可能性があります．

テーブルには，データを変換するデータテーブルのほかに，処理ルーチンを規定するジャンプテーブルがあります．たとえば，アトリビュート形式の変換をテーブルを経由して行う場合には，データテーブルを使用したことになります．これに対して，データに依存した多重分岐をテーブルで行おうという場合には，ジャンプテーブルを利用することになります．

前者はC言語でいえば普通のデータ配列であり，後者は関数へのポインタの配列になります．

たとえば，画面表示を行う場合に，それが半角文字であるか全角文字の1バイト目であるか，はたまたコントロールコードであるかどうかを判別しないといけません．さらに1バイト目がすでに表示された場合に，次のデータがコントロールコードであるかどうかを別に判別しないといけません．

今までのプログラムでは，これらの判別は@if－@else－@e_if－@endifによる多重分岐を使用したり，演算によって連続領域の判別をしたりしてきました．しかし，とくに多重分岐を利用した場合には比較演算と分岐が繰り返し実行されることになり，分岐数が多い場合にオーバーヘッドが大きくなります．

このようなときに，判別されるコードと処理内容とを1：1で記述してあ

るテーブルを参照すれば，いっぺんに処理内容が決定でき，高速です．具体的には，0～255の各キャラクタコードに対して処理内容のアドレスをもつ256ワードの配列を用意しておけばよいのです．そして，キャラクタコードからインデックスを算出し，それを利用して当該処理ルーチンを呼び出すことになります．

しかし，漢字コードの1バイト目が出力されたときと，そうでないときとでは同じコードでも処理内容が異なります．たとえば，41H(A)は漢字コードの1バイト目がすでにあればSJISの2文字目として処理され，そうでなければ半角Aとしてすぐに画面に表示されないといけません．

このようなことを実現するには，複数のテーブルを用意して，そのときの状態に応じて参照するジャンプテーブルを変更すればよい，ということになります．これは，オートマトンを構成するということです．内部ステートと入力コードのマトリクスを作り，そこにジャンプベクタを配し，処理内容に応じて内部ステートを切り替えるのです．

たとえば，81H～9FHまでのコードを受けた場合には，それをワークエリアに格納しステートをSJISの2バイト目状態に切り替え，2バイト目が出力されたら初期状態に戻します．エスケープシーケンスの解釈も同様になります．

さて，では内部ステートはいくつぐらい存在していると考えられるでしょうか．分岐先でステートを示すなんらかの変数を参照して分岐を許すのなら，必要な内部ステートは少なくなります．しかし，今回は基本的にこのような分岐をしないことにします．

まず，通常状態で1つ，SJISの1バイト目を受けた状態が1つ．それからESCを受けて次の文字を待っている状態が1つ．ESCのあとに[を受けて，そのあとの英字（エスケープシーケンスの終了文字）を待っている状態が1つ．さらに，ESC＋＝でロケートをしているときのx／yアドレスを受けるための状態がそれぞれ1つ．計6つの状態が存在していることになります．

もしも，すべてのキャラクタに対応するテーブルを構成したとすると，256＊2＊6で3KB程度のテーブルが必要になります．これは大き過ぎると思われるので，なんとかしないといけません．

今回は，テーブルを小さくするのに，次のような方法を取りました．まず，ある程度以上のコードについては，「共通の処理ルーチン」を呼び出すことがほとんどだということに注目します．たとえば，SJISの1バイト目を受けている状態では，コントロールコード以外を受信した場合はつねに同一の処理が（SJISコードとして正しくても正しくなくても）行われます．ですから，これらすべてにテーブルを割り当てるのは不経済です．

そこで，テーブルの前にテーブルサイズと，それを超えたときのデフォルトの処理アドレスを設定しておくことにします．最初にこれを参照することによって，無駄なテーブルのエントリを削減することができるのです．

リスト10　back4.asm

```
ifdef   ATTR_OFFSET
        public  _disable_cg, _decode_attr
        public  _put_chr, _move_to_screen, _move_between_screen, _store_to_screen
else
        dosseg
        .model  small, pascal
        masm51

        include condx.inc
        include utils.inc

ext_func        disable_cg      <>
ext_func        decode_attr     <>
ext_func        put_chr <mode, adr, _code, attr> <di>
ext_func        move_to_screen  <mode, adr, cnt, srcoff, srcseg> <si di>
ext_func        move_between_screen <srcoff, cnt, dstoff> <si di>
ext_func        store_to_screen <mode, adr, cnt, _code, attr> <si di>

;---------------------------
;io.sysワークエリア
; io.sys workarea
; segment = 71h
;---------------------------
        .data
low_mem struc
row_pos         db      ?
fkey_flag       db      ?
row_last        db      ?
                db      ?
disp_attr       db      ?
                db      ?
                db      ?
                db      ?
wait_flag       db      ?
pad_char        dw      ?
cursor_flag     db      ?
col_pos         db      ?
fill_attr       db      ?
row_top         db      ?
                db      ?
low_mem ends

;---------------------------
;ワークエリア
; work area
;---------------------------

escbuf  db      20 dup (?)              ; escape sequence buffer
escptr  dw      ?                       ; buffer pointer

kbuf    dw      ?                       ; full width char 1st byte

c_attr  dw      0e0h                    ; current byte type attribute

ychr    dw      ?                       ; locate y character

xysave  dw      0                       ; save area for cursor attributes
atrsave dw      0e0h
asave   dw      0

;---------------------------
;ジャンプテーブル
; jump table
;---------------------------

fptr    dw      fbuf_1

; normal condition
        dw      256, _fnull
fbuf_1  label   word
        ; control characters
        dw      _fnull, _flch, _flch, _flch, _flch, _flch, _flch, _fbel
```


ワンラインNEWS
シャープは本体のRAM容量が128Kバイトの電子システム手帳「PA-9550」を発売した．従来の2.5倍の情報を収めることができる．

```
        dw      _fbs,   _fht,   _flf,   _fvt,   _fff,   _fcr,   _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _fsubx, _fesc,  _flch,  _flch,  _frs,   _flch
        ; half width character
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        ; full width entry
        dw      _fnull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull
        dw      _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull
        dw      _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull
        dw      _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull
        ; half width character
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        dw      _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch,  _flch
        ; full width entry
        dw      _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull
        dw      _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull
        dw      _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull
        dw      _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull, _ffull

; sjis 2nd byte entry
        dw      20h, _f2ch
fbuf_2  label   word
        dw      _fnull, _f2ch,  _f2ch,  _f2ch,  _f2ch,  _f2ch,  _f2ch,  _fbel
        dw      _fbs,   _fht,   _flf,   _fvt,   _fff,   _fcr,   _f2ch,  _f2ch
        dw      _f2ch,  _f2ch,  _f2ch,  _f2ch,  _f2ch,  _f2ch,  _f2ch,  _f2ch
        dw      _f2ch,  _f2ch,  _fsubx, _f2ch,  _f2ch,  _f2ch,  _frs,   _f2ch

; escape first code
        dw      60h, _fnorm
fbuf_3  label   word
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fcls,  _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _floc2, _fnorm, _fnorm
        ; 40h
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fdownl, _fcrlf, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fupl,  _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fesc2, _fnorm, _fnorm, _fnorm, _fnorm

; escape sequence check
        dw      80h, _fnorm
fbuf_4  label   word
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnext, _fnext, _fnext, _fnext, _fnext, _fnext, _fnext, _fnext
        dw      _fnext, _fnext, _fnext, _fnext, _fnext, _fnext, _fnext, _fnext
        dw      _fnext, _fnext, _fnext, _fnext, _fnext, _fnext, _fnext, _fnext
        dw      _fnext, _fnext, _fnext, _fnext, _fnext, _fnext, _fnext, _fnext
        ; 40H
        dw      _fnorm, _fup,   _fdown, _fright, _fleft, _fnorm, _fnorm, _fnorm
        dw      _floc,  _fnorm, _feos,  _feol,  _fins,  _fdel,  _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm
        ; 60H
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _floc,  _fnorm
        dw      _fextl, _fnorm, _fnorm, _fnorm, _fext2, _fcolor, _fread, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fsavex, _fnorm, _frest, _fnorm, _fnorm
        dw      _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm, _fnorm

; locate 1st char
        dw      0, _flocc1
fbuf_5  label   word
        dw      _flocc1

; locate 2nd char
        dw      0, _flocc2
fbuf_6  label   word
        dw      _flocc2

;-----------------------------
;割り込みエントリ
; interrupt entry
;-----------------------------
        .code

_my_int_func:
        ; main interrupt routine
        sti
        push    ds
        push    es
        push    ax
        push    bx
        push    cx
        push    dx

        ; set data segment
        mov     bx, _DATA
        mov     ds, bx

        ; call according to jump table
        mov     bx, fptr
        mov     ah, 0
        cmp     [bx-4], ax
        @if     be
                ; default routine
                push    ax
                call    [bx-2]
        @else
                ; calc and reference table
                add     bx, ax
                add     bx, ax
                push    ax
                call    [bx]
        @endif

        ; CG termination
        disable_cg <>

        pop     dx
        pop     cx
        pop     bx
        pop     ax
        pop     es
        pop     ds
        iret

;-----------------------------
;ユーティリティルーチン
; utility routines
;-----------------------------

        extrn   my_vect:dword

chain   macro
        pushf                                   ;; emulate int call
        cli
        call    my_vect
        endm

func    get_pos <>
        ; get video address in bx, row/col in ch/cl
        ; last_line in dx
        push    ds
        mov     bx, 71h
        mov     ds, bx
        mov     dh, ds:[row_pos]
        mov     dl, ds:[col_pos]

        ; bx = 160*dh
        mov     bl, dh
        mov     bh, 0
        mov     cl, 5
        shl     bx, cl
        mov     cx, bx
        add     bx, bx
        add     bx, bx
        add     bx, cx

        ; calc screen address
        mov     ch, 0
        mov     cl, dl
        add     bx, cx
        add     bx, cx
        mov     cx, dx

        ; screen lines set in dx
        mov     dh, 0
        mov     dl, ds:[row_last]

        pop     ds
        ret
endf

func    set_pos <pos, vadr>
        push    ds
        mov     ax, 71h
        mov     ds, ax

        ; store cursor position
        mov     ax, pos
        mov     ds:[row_pos], ah
        mov     ds:[col_pos], al

        ; special for right edge cursor
        mov     dx, vadr
        cmp     al, 80
        @if     ae
                dec     dx
                dec     dx
        @endif

        ; cursor move bios call
        mov     ah, 13h
        int     18h
        pop     ds
        ret
endf

func    set_attr        <attr>
        push    ds
        mov     ax, 71h
        mov     ds, ax

        ; set same attribute into low memory
        mov     ax, attr
        mov     ds:[disp_attr], al
```

その他のテーブルの作成法

今回は採用しませんでしたが，より美しい解決の仕方もあります．各ステートとそのときの処理アドレスとの6組のペアを見てみると，多くの文字でこれらがまったく同一になっていることに気がつきます．たとえば，文字コードが81Hと82Hでは完全に同一です．83Hも同一です．しかし，41H(A)と42H(B)では，エスケープシーケンスの解釈のところで多少の差異があります．

このような差異がまったくない文字をグループにすると，それはさして大きくない数になると思われます．このグループの数が仮に20個あったとします．まず，1番目の256バイトのテーブルで，文字コードからグループコードへのマッピングを行います．そして，現在のステートに応じたグループコードから飛び先へのテーブルを参照すれば，処理内容が決定できるのです．つまり，テーブルを二段がまえにするのです．

すると，最初のテーブルが256バイト，2番目のテーブルは20＊5＊6で600バイトとなり，トータルで1KBにもなりません．テーブルの大きさは，大幅に削減されるのです．

さらに，この最初のテーブルはほかの部分(たとえば，backcol. asmの中でSJISコードの判定をしている部分)で共用するようにできる可能性があります．単なるジャンプテーブルだと，そういう共用ができにくくなります．

問題は，グルーピングをするのが多少厄介だということです．誤りも比較的発見しにくくなりますし，既存のテーブルでの分岐先が増える(たとえば，新しいエスケープシーケンスをサポートする）とグループが増えることになるので，いろいろと変更すべき部分が生じてきます．そのため，メンテナンスは多少やりにくくなります．

テーブルサイズを削減するためには，直接飛び先を記述せずに，飛び先のインデックスを記述し（これはたぶん1バイトでたりるだろうから)，飛び先テーブルをこのインデックスで参照することによって処理内容を決定するという手法もとれます．やはり2段処理になりますが，メンテナンスの点ではグルーピングをするよりも楽ですし，テーブルサイズもグルーピングほどではありませんが，元の半分程度まで削減できます．

このように，テーブルの作り方もいろいろありますが，うまく使えば高速な処理を行うことができます．もっとも，コードのパーサだけ（メモリサイズを犠牲にして）高速にしても，ほんとうはあまり意味がありません．処理内容の重さ，たとえば画面出力ルーチンのスピードに応じた速度を出すだけで十分なのです．ここを高速化するなら，画面出力ルーチンもより一層の高速化を計る必要があるでしょう．

プログラムの解説

back4.asm

では，今回のプログラムを見ていきます．最初，今回のプログラム全体とback2. asm を backcol. asm にincludeさせてアセンブルすると，TASMのバージョンによって不安定な動作が生じました．MASMでは問題なかったのですが，おそらくマクロなどの展開結果が大きすぎたためと判断し，分割アセンブルをすることにしました．

その場合，public宣言やext_func宣言が必要になるので，ifdefで今回のback4. asmを二重に使えるようにしました．back2といっしょに使用される場合には，先頭のpublic宣言のみが有効になり，そうでない場合にはext_funcを併用し，完結したプログラムになるようになっています．

これらの宣言の次にあるのは，io. sysの中にあるカーソル関係のワークエリアの記述です．ここではstrucを利用してアドレスオフセットを決定しています．このワークエリアは，セグメント71Hから用意されています．

本来，これらのカーソルアドレスやアトリビュートはプログラム内部にもっていたほうが簡単ですし，高速になります．しかし，前記したようにこのプログラムではすべての出力をトラップすることはできません．ですから，最低限のワークエリアは共有しておくほうが無難です．このようにしておくことによって，バックカラーこそつきませんがINT DCHで文字の出力ができるでしょう．

ワークエリアには，カーソルの行／桁の位置のほかに，スクロール範囲，各種アトリビュートとクリアキャラクタ，ファンクションキーフラグなどがあります．今回サポートするのは，このカーソル位置とスクロール下限のみにします．つまり，

・カーソルのon／offはサポートされていない
・画面のクリアキャラクタは参照されていない（つねに半角スペース)
・スクロール範囲の上限はサポートされていない（つねに1行目から)
・表示アトリビュートとクリアアトリビュートは同一
・ファンクションキーはサポートされていない

```
        mov     ds:[fill_attr],al

        pop     ds
        ret
endf

func    roll_up <top,tail,line>,<si di>
        ; calc top vadr
        mov     ax,top
        mov     ah,160
        mul     ah
        mov     si,ax

        ; calc source offset
        mov     ax,line
        mov     ah,160
        mul     ah
        mov     dx,ax

        ; calc lines to copy
        mov     di,tail
        sub     di,top
        inc     di
        sub     di,line
        @if     ae
                ; copy loop
                @begin
                        push    dx
                        mov     ax,si
                        add     ax,dx
                        move_between_screen <ax,80,si>
                        pop     dx

                        ; advance pointer
                        add     si,160
                        dec     di
                @until  z
                mov     di,line
        @else
                add     di,line
        @endif

        ; clear bottom lines
        or      di,di
        @if     nz
                @begin
                        store_to_screen <0,si,80,' ',c_attr>
                        add     si,160
                        dec     di
                @until  z
        @endif

        ret
endf

func    roll_down       <top,tail,line>,<si di>
        ; calc tail vadr
        mov     ax,tail
        mov     ah,160
        mul     ah
        mov     si,ax

        ; calc source offset
        mov     ax,line
        mov     ah,160
        mul     ah
        mov     dx,ax

        ; calc lines to copy
        mov     di,tail
        sub     di,top
        inc     di
        sub     di,line
        @if     ae
                ; copy loop
                @begin
                        push    dx
                        mov     ax,si
                        sub     ax,dx
                        move_between_screen <ax,80,si>
                        pop     dx

                        ; advance pointers
                        sub     si,160
                        dec     di
                @until  z
                mov     di,line
        @else
                add     di,line
        @endif

        ; clear top lines
        or      di,di
        @if     nz
                @begin
                        store_to_screen <0,si,80,' ',c_attr>
                        sub     si,160
                        dec     di
                @until  z
        @endif

        ret
endf

func    adjust_top <>
        get_pos <>

        ; check right edge
        cmp     cl,80
        @if     b
                ; not on right edge, quick return
                ret
        @endif

        ; advance cursor row (column becomes 0)
        mov     cl,0
        inc     ch
        cmp     ch,dl
        @if     a
                ; last line, scroll up
                dec     ch
                sub     bx,160
                push    cx
                push    bx

                roll_up <0,dx,1>

                pop     bx
                pop     cx
        @endif

        ret
endf

func    atoi    <p>
        ; convert ascii to integer
        ; 0 is default value
        mov     ax,0
        mov     bx,p

        @begin
                mov     cl,[bx]
                cmp     cl,'0'
                @if     b
                        @break
                @endif
                cmp     cl,'9'
                @if     a
                        @break
                @endif

                mov     ah,10
                mul     ah

                sub     cl,'0'
                add     al,cl

                inc     bx
        @again

        ; bx points first non digit character
        ret
endf

;----------------------------
;各処理ルーチン
; parsing routines
;----------------------------

func    fnull   <chr>
        ; nothing
        ret
endf

func    fnorm   <chr>
        ; reset function table pointer
        mov     fptr,offset fbuf_1
        ret
endf

func    fich    <chr>,<si di>
        adjust_top <>
        mov     si,cx
        mov     di,bx

        ; write character
        put_chr <0,di,chr,c_attr>

        ; cursor position set
        inc     si
        inc     di
        inc     di
        set_pos <si,di>

        ret
endf

func    fbel    <chr>
        ; invoke old int29
        mov     ax,chr
        chain
        ret
endf

func    fbs     <chr>
        get_pos <>

        ; back move
        dec     bx
        dec     bx
        sub     cl,1
        @if     b
                ; left edge, move prev. line
                mov     cl,79
                sub     ch,1
                @if     b
                        ; left-up corner, never move
                        mov     cx,0
                        inc     bx
                        inc     bx
```

ワンラインNEWS
日本電気はビジネス向けパソコン「N5200」の新機種を発売した。「モデル03sx」はディスプレイ一体型でCPUにインテル社の「80386SX（16MHz）」を搭載している。

というような制限があります．

ここで，ファンクションキーについて補足しておきます．ファンクションキーがOFFである状態では，画面全部がスクロール領域になるのでとくに問題はありません．問題はONのときです．このときには，今回のプログラムはあたかも「ファンクションキーが画面に存在しない」ように扱います．つまり，クリアが実行されてもファンクションキーについてはノータッチです．ですから，バックカラーをつけてクリアをしても，ファンクションキーの部分には色がつきません．

これは，ファンクションキーの表示部分が単なるINT 29Hのハンドラを超えたレベルのものだからです．たとえば，これらの切り替えはctrl+f7で行えます．つまり，キー入力も監視しておかないと，完全な表示はできません．また，表示する内容もio. sysの内部を探さないと見つけることができません．ある意味で，今回のプログラムは「サンプル」ですから，これらの問題についてはオミットしています．もっとも，たとえばクリアが実行されたときに，ファンクションキーが表示されている場合でもそのバックカラーだけ設定する，という方法を用いることはできます．このほうが，動作としては自然かもしれません．

escbuf／escptr

次に，プログラムのローカル変数領域があります．escbuf／escptrはエスケープシーケンスの内容を格納するものです．ANSIのエスケープシーケンスでは，esc+[から英文字までの数値列などに応じて処理を決める必要があります．これを格納しておくためのバッファです．

kbuf／c_attr／ychr

kbufはSJISの1バイト目を保存するもの，c_attrは現在のアトリビュート（バイト形式）を保存しておくもので，ychrはesc+=のタイプのカーソル移動を行うときのY座標を示すものです．

xysave／atrsave／asave

xysave／atrsave／asaveは，それぞれ，カーソル座標と現在のアトリビュートを保存する領域で，エスケープシーケンスによってカーソル状態の保存／復元を行うときに使用します．

fbuf_1／fbuf_2

その次に，fptrとあるのは現在のジャンプテーブルの先頭を示すポインタです．それから，先に述べた6種類の内部状態に応じたテーブルが用意されています．最初のテーブルは通常状態のもので，コントロールキャラクタ，半角キャラクタ，SJISの1バイト目を処理します．

内容は直接関数のエントリを示すワードサイズのアドレスで，func使用の関係から頭に「_」が付いています．f1chは半角の1文字出力，ffullはSJISの1バイト目，fcrなどはそれぞれコントロールコードの処理を示します．このように256キャラクタの中に処理内容の異なるものが混在しているときには，テーブリングは比較的有効に使用できます．テーブル先頭の大きさを示す領域は，256文字全部有効となっています．

次のテーブルは，SJIS1バイト目が出力されたときに2バイト目を処理するためのものです．コントロールキャラクタの一部を除いて，つねにf2chが呼び出されます．これは全角文字を表示する関数で，それが実行されると関数テーブルポインタはfbuf_1に復帰することになります．このテーブルは，コントロールキャラクタ以外はすべてf2chを呼び出すので，先頭でその大きさとデフォルトの飛び先f2chを定義しています．

fbuf_3

その次のfbuf_3は，エスケープコードが出力されたときの次の文字を解釈するためのテーブルです．fnormはエスケープシーケンスを無視し，テーブルポインタを復元するためのもので，大部分はそのようになっています．エスケープ文字に特別のキャラクタ，たとえば「*」であるとか「[」が続いたときにだけエスケープシーケンスを解釈する必要があるのです．

このうちesc+*やesc+Aなど1文字で完結するシーケンスは，直接その処理が呼ばれます．esc+=はそのあとの2文字を待たないといけないので，やはり独自の処理をします．esc+[であった場合には，英文字がくるまで待たないといけないので，そのためのテーブルをfesc2でセットします．

fbuf_4

fbuf_4はANSIのエスケープシーケンスを解釈しているときのテーブルで，基本的に英字がくるのを待ちます．数字などであれば，これをバッファに格納するだけでテーブルポインタは変更しません．英文字がきたら，それをもとにバッファの内容を解析し，テーブルポインタをfbuf_1に戻します．このときの解釈ルーチンは，英文字に依存するのでそれを直接テーブルで記述することができます．

```
                @endif
        @endif

        ; and set position
        set_pos <cx,bx>
        ret
endf

func    fht     <chr>,<si>
        get_pos <>

        ; calc number of spaces to output
        and     cl,7
        mov     ch,0
        mov     si,8
        sub     si,cx

        @begin
                ; output space
                flch <' '>
                dec     si
        @until  z
        ret
endf

func    flf <chr>
        get_pos <>

        ; move to next line
        inc     ch
        add     bx,160
        cmp     ch,dl
        @if     a
                ; last line, scroll up
                dec     ch
                sub     bx,160
                push    cx
                push    bx
                roll_up <0,dx,1>
                pop     bx
                pop     cx
        @endif

        ; and set cursor
        set_pos <cx,bx>
        ret
endf

func    fvt     <chr>
        get_pos <>

        ; move previous line
        or      ch,ch
        @if     nz
                ; not on first line, move
                dec     ch
                sub     bx,160

                set_pos <cx,bx>
        @endif
        ret
endf

func    fff <chr>
        adjust_top <>

        ; right move
        inc     cl
        inc     bx
        inc     bx
        set_pos <cx,bx>
        ret
endf

func    fcr <chr>
        get_pos <>

        ; clear column position
        sub     ax,ax
        xchg    al,cl
        sub     bx,ax
        sub     bx,ax
        set_pos <cx,bx>
        ret
endf

func    frs <chr>
        ; move to home
        mov     cx,0
        mov     bx,cx
        set_pos <cx,bx>
        ret
endf

func    fsubx <chr>
        ; clear screen
        get_pos <>

        ; calc displaied nubmers
        mov     ax,80
        mul     dx
        store_to_screen <0,0,ax,' ',c_attr>

        frs <chr>
        ret
endf

func    fesc    <chr>
        ; set escape sequence parsing table
        mov     fptr,offset fbuf_3
        ret
endf

func    ffull   <chr>
        ; store SJIS first byte
        mov     ax,chr
        mov     kbuf,ax

        mov     fptr,offset fbuf_2
        ret
endf

func    f2ch    <chr>,<si di>
        adjust_top <>

        cmp     cl,79
        @if     z
                ; odd one char, output space
                flch <' '>
                adjust_top <>
        @endif

        mov     si,cx
        mov     di,bx

        ; write 2 characters
        put_chr <0,di,kbuf,c_attr>
        inc     si
        inc     di
        inc     di
        put_chr <0,di,chr,c_attr>
        inc     si
        inc     di
        inc     di
        set_pos <si,di>

        ; clear buffer
        mov     kbuf,0
        mov     fptr,offset fbuf_1
        ret
endf

func    fcls    <chr>
        ; clear screen and move to home position
        fsubx <ax>
        fnorm <ax>
        ret
endf

func    floc2   <chr>
        ; wait y position
        mov     fptr,offset fbuf_5
        ret
endf

func    fdownl  <chr>
        ; same as line feed, reset table pointer
        flf <ax>
        fnorm <ax>
        ret
endf

func    fcrlf   <chr>
        fcr <ax>
        flf <ax>
        fnorm <ax>
        ret
endf

func    fupl    <chr>
        get_pos <>

        ; up one line, if need scroll down
        or      ch,ch
        @if     z
                roll_down <0,dx,1>
        @else
                dec     ch
                sub     bx,160
        @endif

        set_pos <cx,bx>
        fnorm <ax>
        ret
endf

func    fesc2   <chr>
        ; pointer/buffer initialize
        mov     escptr,offset escbuf
        mov     escbuf,0

        ; table pointer set
        mov     fptr,offset fbuf_4
        ret
endf

func    fnext   <chr>
        ; receive character, and save it into buffer
        mov     bx,escptr
        mov     ax,chr
        mov     [bx],al
        inc     bx

        ; terminate string
        mov     byte ptr [bx],0

        ; check buffer length
        cmp     bx,offset escbuf+20
        @if     ae
                dec     bx
```

ワンラインNEWS
九州松下電器は，世界最高の記憶密度をもつHDDを開発した．このHDDは磁気ヘッドとディスクの間の間隔が0.1ミクロンと従来の1/2〜2/3の間隔となっている．

fbuf_5／fbuf_6

fbuf_5／fbuf_6は，それぞれesc＋＝のカーソル移動シーケンスの1バイト目と2バイト目を受けるときのテーブルです．これらはサイズ0のテーブルですが，ychrの値を状態として使用することで共用させることも不可能ではありません．

_my_int_func

そのあとに，実行コードが続きます．以前に掲載したresident.asmと共用するので，割り込みルーチンは_my_int_funcとなります．このルーチンでは，セグメント・データレジスタを保存し，DSにデータセグメントを設定したあとALのキャラクタコードによって分岐します．ただし，最初にテーブルサイズとの比較を行い，テーブルサイズを超えていればデフォルトのルーチンをコールします．そうでなければ，テーブル参照を行い，そこをコールします．

呼び先は，すべてキャラクタコードを引数と取ることにしているので，AXレジスタをプッシュしてから呼び出しています．固定領域にコードを保存してから呼び出したほうが効率は上がります．

ここでわかるように，複雑な分岐をする場合には，テーブルを使うと実行コードをたいへんに簡潔に記述できます．

chain

それから，まずいくつかの補助関数が定義されています．最初のchainはマクロで，元のINT 29Hルーチンを呼び出そうというものです．pushf＋call far命令でintをエミュレートします．このために，residentで保存されているアドレスが必要になります．呼び出し前には，cliを実行しておくのが無難でしょう．

get_pos

get_posは，セグメント71Hにあるio.sysのワークの内容を取得するものです．カーソル位置はch／clに，スクロール下限はDXに，BXにはカーソル位置の画面上のアドレスが格納されます．BXはCXの値から算出しています．

ここで160倍というかけ算が必要になるのですが，かけ算命令はあまり使用したくないので，たし算だけでまかなっています．こうすればAXレジスタがフリーになります．比較的2^nに近い値のかけ算は，このようにたし算だけで行うと多少ですが高速になることがあります．ただし，スピードが気にならない場合は，かけ算命令を使うほうが簡潔でしょう．

set_pos

set_posは，その逆にカーソルワークエリアとカーソル位置の設定を行うものです．ここで注意するのは，PC-9801のデフォルトのドライバでは，80桁目に文字を出力してもカーソルは次の行に移動しないということです．IBM PCなどでは，移動するのが普通です．

このために，現在の桁数が80になっている場合には，カーソル位置を設定するための画面上のアドレスを調整し，その行にカーソルが納まるようにします．カーソル設定は，前回と同様にbios呼び出しを使用します．

set_attr

set_attrは，アトリビュートのみを設定するものです．前記ワークエリアの2つのアトリビュートの双方に同一の値を設定します．

roll_up／roll_down

roll_up／roll_downは，画面上の任意の行範囲で任意行数のスクロールを実行するものです．どうせ時間のかかるルーチンですから，アドレス計算にはmulなどを使用しています．基本的には，前回のウィンドウ内のスクロールと同様に，1行単位で画面内のコピーを実行します．ただ，桁の考慮が不要なので，実際には1回のmove_between_screenの呼び出しをするだけでも十分です．最後に，スクロールインした部分をstore_to_screenでクリアします．クリアキャラクタは半角スペースを使用しています．

adjust_top

adjust_topは，文字出力をする場合の調整ルーチンです．前記のように桁位置が80であり，かつ最下行である場合には，次の文字を出力する「前」にスクロールをする必要があります．ですから，これをチェックし，必要な場合にはスクロールを実行し，レジスタの値を調整します．画面に文字を出力しているときのスクロールは，基本的に文字の出力後ではなく出力前に行います．

atoi

atoiは，エスケープシーケンス解釈のための数値解釈ルーチンです．BXレジスタは非数値文字を指しますので，esc＋［...mのように複数の数値を解釈する必要がある場合にはこのレジスタの値＋1を次の引数にすることができます．

```
        @endif
        mov     escptr,bx
        ret
endf

func    flocc1  <chr>
        get_pos <>

        ; adjust position
        mov     ax,chr
        sub     ax,' '

        ; check range
        cmp     ax,dx
        @if     ae
                mov     ax,dx
        @endif

        ; save it into variable
        mov     ychr,ax

        ; next, wait x position
        mov     fptr,offset fbuf_6
        ret
endf

func    flocc2  <chr>
        ; modify x position
        mov     ax,chr
        sub     ax,' '
        cmp     ax,80
        @if     ae
                mov     ax,79
        @endif

        ; load positions into register
        mov     cl,al
        mov     ax,ychr
        mov     ch,al

        ; calculate screen address
        mov     ah,160
        mul     ah
        mov     dl,cl
        mov     dh,0
        add     ax,dx
        add     ax,dx
        mov     bx,ax

        set_pos <cx,bx>
        fnorm <ax>
        ret
endf

func    fup     <chr>
        ; parse lines to up
        atoi <<offset escbuf>>

        get_pos <>

        ; check upper edge
        cmp     al,ch
        @if     ae
                mov     al,ch
        @endif

        ; modify screen address
        sub     ch,al
        mov     ah,160
        mul     ah
        sub     bx,ax

        ; cursor move
        set_pos <cx,bx>

        fnorm <ax>
        ret
endf

func    fdown   <chr>
        atoi <<offset escbuf>>
        get_pos <>

        ; down move, check lower edge
        mov     dh,dl
        sub     dh,ch
        cmp     al,dh
        @if     ae
                mov     al,dh
        @endif

        ; calculate screen address
        add     ch,al
        mov     ah,160
        mul     ah
        add     bx,ax

        set_pos <cx,bx>

        fnorm <ax>
        ret
endf

func    fright  <chr>
        atoi <<offset escbuf>>
        get_pos <>

        ; check right edge
        mov     dl,79
        sub     dl,cl
        cmp     al,dl
        @if     ae
                mov     al,dl
        @endif

        ; correct screen address
        add     cl,al
        mov     ah,0
        add     bx,ax
        add     bx,ax

        set_pos <cx,bx>

        fnorm <ax>
        ret
endf

func    fleft   <chr>
        atoi <<offset escbuf>>
        get_pos<>

        ; check left edge
        cmp     al,cl
        @if     ae
                mov     al,cl
        @endif

        ; correct screen address
        sub     cl,al
        mov     ah,0
        sub     bx,ax
        sub     bx,ax

        set_pos <cx,bx>

        fnorm <ax>
        ret
endf

func    floc    <chr>
        ; parse first line
        atoi <<offset escbuf>>

        ; save pointer
        push    bx

        ; check range
        get_pos <>
        pop     bx
        cmp     ax,dx
        @if     ae
                mov     ax,dx
        @endif

        ; save it
        mov     ychr,ax

        ; convert x position
        mov     ax,0
        cmp     byte ptr [bx],';'
        @if     z
                inc     bx
                atoi <bx>
        @endif

        ; add offset
        add     ax,' '

        ; and move
        flocc2 <ax>
        ret
endf

func    feos    <chr>
        atoi <<offset escbuf>>
        get_pos <>
        cmp     al,0
        @if     z
                ; clear to eos
                mov     al,dl
                sub     al,ch
                inc     al
                mov     ah,80
                mul     ah
                mov     ch,0
                sub     ax,cx
                store_to_screen <0,bx,ax,' ',c_attr>
        @else
        cmp     al,1
        @e_if   z
                ; clear from top
                mov     al,ch
                mov     ah,80
                mul     ah
                mov     ch,0
                inc     cx
                add     ax,cx
                store_to_screen <0,0,ax,' ',c_attr>
        @else
        cmp     al,2
        @e_if   z
                ; clear full screen
                fcls <ax>
        @endif
        fnorm <ax>
        ret
endf

func    feol    <chr>
        atoi <<offset escbuf>>
```

fnull／fnorm

そのあとにテーブル参照で呼び出される各処理ルーチンが記述されています．

fnullはなにもしないルーチン，fnormはテーブルポインタを初期化するルーチンです．

f1ch

f1chは半角１文字を表示するルーチンで，まずadjust_topでアドレスを得たあと，put_chrで文字を画面に書き込みます．ポインタを進ませてからset_posで設定をします．文字表示後にはスクロールは起こらないので，カラムアドレスは単にインクリメントするだけでかまいません．

fbel

fbelはビープルーチンで，単に元のINT 29Hを呼び出します．

fbs〜fsubx

このあとにあるfbs〜fsubxまでは，それぞれコントロールコードに対応するルーチンで，主としてカーソル移動を行います．get_posまたはadjust_topによってカーソル位置を取得し，指定の方向にカーソル移動したあと，それを設定します．また，flfなど必要に応じてスクロールアップ処理を行っています．fsubxがxを付加されているのは，fsubが既存の命令と重なるためです．

fesc

エスケープシーケンスは，fescで処理されます．最初は，テーブルポインタにfbuf_3を設定するだけです．

ffull／f2ch

ffullはSJISの１バイト目で，受けたコードをchrに保存したあと，テーブルポインタを切り替えます．f2chは，この切り替わったテーブルポインタ経由で呼ばれるもので，まず右端のチェックをし，必要ならスペースを１個出力します．それから連続してSJISコードを書き出します．テーブルポインタは，初期状態に復元します．

fcls／fdown

fcls／fdownなどは，esc＋１文字で呼び出されるエスケープシーケンスの処理です．カーソル移動などの内容は，ほぼコントロールコードで処理されたものと同様です．

esc＋＝の場合は，テーブルポインタをfbuf_5に設定し，次のコードの受信を待ちます．まずflocc1が呼ばれ，次にflocc2が呼ばれます．前者はｙアドレスをychrに保存するだけで，後者によって実際にカーソルの移動が行われます．テーブルポインタの復元も同時に行われます．

esc＋［のANSIシーケンスであった場合，シーケンスが終了するまでの待ち状態に入ります．これはfesc2で行われ，各種ポインタの初期化も実行されます．シーケンスの受信はfnextで行われ，オーバーフローチェックに備えます．

fup

このあとのfupからのルーチンは，エスケープシーケンスが完了したあとの解釈ルーチンです．カーソル移動に関するルーチンは，基本的に一文字移動のルーチンと同じですが，移動文字数を解釈する必要があります．これには，atoiを使用してescbufの内容を変換する必要があります．このとき，引数には「offset　escbuf」を渡す必要があるので<>でもう一重よけいに括ってやります．あるいは，mov命令やlea命令でアドレスをレジスタに読み込んだあと，それを引数に取ってもかまわないのですが，ローカル変数をアクセスする場合には，lea命令を使ってアドレスを取得する必要があります(ただし，割り込みルーチンではDS！＝SSだから，安易にローカル変数のアドレスを取るとたいへんなことになります)．変換された値はAXに返されるので，これに応じてカーソル移動文字数を計算します．

floc

flocでは，２つの数値を解釈する必要があります．前記のようにBXに非数字のアドレスが返されるので，これが「；」であった場合には１文字スキップすることにより次の数値の変換が可能です．実際の移動はflocc2を呼び出しています．

feos／feol

feos／feolは，画面・行の部分クリアです．この場合も引数に依存して処理内容を決めています．基本的にはクリアする文字数を計算し，store_to_screenを呼び出しています．

fins／fdel

fins／fdelは行挿入・削除のコマンドですが，実際には画面の部分スクロールという形で実現されます．必要な開始行数とスクロールすべき行数を計算し，roll_down／roll_upを呼び出します．

fext1／fext2

PC-9801の機能，たとえばカーソル

```
	get_pos <>
	cmp	al,0
	@if	z
		; clear to eol
		mov	ax,80
		mov	ch,0
		sub	ax,cx
		store_to_screen <0,bx,ax,' ',c_attr>
	@else
	cmp	al,1
	@e_if	z
		; clear from tol
		mov	ch,0
		sub	bx,cx
		sub	bx,cx
		inc	cx
		store_to_screen <0,bx,cx,' ',c_attr>
	@else
	cmp	al,2
	@e_if	z
		; clear full line
		mov	ch,0
		sub	bx,cx
		sub	bx,cx
		store_to_screen <0,bx,80,' ',c_attr>
	@endif

	fnorm <ax>
	ret
endf

func	fins	<chr>,<si di>
	atoi <<offset escbuf>>
	mov	si,ax

	; move to left edge
	get_pos <>
	mov	di,dx
	mov	al,cl
	mov	ah,0
	sub	bx,ax
	sub	bx,ax
	mov	cl,0
	set_pos <cx,bx>

	; and roll down
	mov	al,ch
	mov	ah,0
	roll_down <ax,di,si>

	fnorm <ax>
	ret
endf

func	fdel	<chr>,<si di>
	atoi <<offset escbuf>>
	mov	si,ax

	; move to left edge
	get_pos <>
	mov	di,dx
	mov	al,cl
	mov	ah,0
	sub	bx,ax
	sub	bx,ax
	mov	cl,0
	set_pos <cx,bx>

	; and scroll up
	mov	al,ch
	mov	ah,0
	roll_up <ax,di,si>

	fnorm <ax>
	ret
endf

func	fext1	<chr>
	; not implemented yet
	fnorm <ax>
	ret
endf

func	fext2	<chr>
	; not implemented yet
	fnorm <ax>
	ret
endf

	.data
	; argument-attribute convert table
color_map_table_1	db	0,40h,80h,0c0h,20h,60h,0a0h,0e0h
color_map_table_2	db	0,2,4,6,1,3,5,7
	.code

func	fcolor	<chr>,<si>
	mov	si,offset escbuf-1
	@begin
		; parse value
		inc	si
		atoi <si>
		mov	si,bx

		; check argument
		mov	cx,ax
		mov	ax,c_attr
		cmp	cx,0
		@if	z
			; init value
			mov	al,0e0h
		@else
		cmp	cx,4
		@e_if	z
			; underline
			or	al,10h
		@else
		cmp	cx,5
		@e_if	z
			; blink
			or	al,8
		@else
		cmp	cx,7
		@e_if	z
			; reverse
			mov	ah,al
			and	al,18h
			mov	ch,ah
			mov	cl,5
			shl	ch,cl
			or	al,ch
			shr	ah,cl
			or	al,ah
		@else
		sub	cx,30
		cmp	cx,7
		@e_if	be
			; for color set
			mov	bx,cx
			and	al,not 0e0h
			or	al,color_map_table_1[bx]
		@else
		sub	cx,10
		cmp	cx,7
		@e_if	be
			; back color set
			mov	bx,cx
			and	al,not 7
			or	al,color_map_table_2[bx]
		@endif

		; save attribute
		mov	ah,0
		mov	c_attr,ax

		; expand it
		decode_attr <>

		; and save it into low area
		set_attr <ax>

		; check more values exist
		cmp	byte ptr [si],';'
	@until	nz

	fnorm <ax>
	ret
endf

func	fread	<chr>
	; read cursor position, pass all sequence to old int 29
	mov	al,1bh
	chain
	mov	al,'['
	chain
	mov	al,'6'
	chain
	mov	al,'n'
	chain
	fnorm <ax>
	ret
endf

func	fsavex	<chr>
	get_pos <>

	; save condition
	mov	xysave,cx
	mov	asave,bx
	mov	ax,c_attr
	mov	atrsave,ax

	fnorm <ax>
	ret
endf

func	frest	<chr>
	; restore condition
	mov	cx,xysave
	mov	bx,asave
	set_pos <cx,bx>

	mov	ax,atrsave
	mov	c_attr,ax

	mov	ax,0
	mov	xysave,ax
	mov	asave,ax

	fnorm <ax>
	ret
endf

public	_my_int_func

endif
	end
```

ワンラインNEWS
東芝は8Kビットの連続したデータを1ビット当たり9nsで読み出すことのできる64MビットDRAMを開発した。64MビットDRAMではこれまで15nsが世界最速だった。

のon／offやファンクションキーのon／offなどのエスケープシーケンスは，そのあとのfext1／fext2に該当します．ただし，ここではこれらのシーケンスはサポートしていません．単にシーケンス全体を元のINT 29Hドライバに流してやるのがいちばん簡単かもしれません．

fcolor

次のfcolorは，画面のフォア・バックカラーを設定するルーチンです．その前のカラーマップテーブルは，エスケープシーケンスコードと，それに対応するアトリビュート値を変換するためのテーブルです．

このシーケンスは複数の数値を解釈する必要があるため，ループを構成しています．0／4／5などは単に現在あるアトリビュート（バイト形式）に特定のビット操作をするだけです．バイト形式のアトリビュートを使用しているため，たとえばシークレットアトリビュートなどのサポートはありません．

7はリバース処理ですが，ここではフォアとバックのアトリビュートを入れ替えています．さらに30～37と40～47はそれぞれフォア・バックアトリビュートの設定ですので，ベース値を引いて変換テーブルを経由してアトリビュートの変換をします．最後にこれを保存し，またワード形式に変換したものの物理的なアトリビュート部分をio. sysのワークエリアに設定します．

fread

いちばん最後のほうのfreadは，カーソル位置をコンソール読み込みで返すためのシーケンスですが，単にINT 29Hに流しています．これは，今回のプログラムがコンソールの入力デバイスドライバをもっていないために，これと通信することができないからです．

fsavex／frest

fsavex／frestはカーソル保存／復元の手続きですが，とくにむずかしい点はないでしょう．

以上で，プログラムの概要の説明を終ります．

プログラムの作成

分割して，アセンブルする

このプログラムを実行ファイルにするためには，まずbaclcol. asmにback2. asmとback4. asmがincludeされるように書き換えます．それをアセンブルしてbackcol. objを作成します．ここで，back4. asmは先頭部分のみ有効になっていることに注意してください．

さらに，back4. asm自体を単体でアセンブルしてback4. objを作成します．その後にbackcol. objとback4. obj，そしてresident. objとをリンクします．前に述べたように，resident. objは必ず「最後に」リンクされる必要があります．

こうしてできた実行ファイルをresident. exeとすると，起動は

```
A:¥> resident -i29
```

のようになります．これによって明示的にINT 29Hのルーチンをトラップするわけです．場合によってはほかのベクタをトラップし，その割り込みを経由して画面アクセスすることも不可能ではありませんが，その場合は元のINT 29Hを呼ぶファンクションを殺すか出力しないように気をつける必要があります．

これを常駐させておけば，画面の表示速度は多少遅くなりますが，通常のdos操作で画面出力をそのまま行うことができます．また，バッチファイルなどのecho命令でエスケープシーケンスを埋めこんでおけば，バックカラーの入ったメッセージを出すことができますし，バックカラーも文字に同期してスクロールアップさせることができます．不要になれば，resident -rで解除することもできます．

また，基本的に元のINT 29Hは呼ばれなくなるので，これをトラップするもの，たとえばFepやxscriptなどの常駐ものよりも先にメモリ内に常駐させる必要があります．

ビデオシステム編終了

できれば，デバイスドライバとして完成を

今回のプログラムは，dosの出力の一部のみをトラップしたものですから，完全ではありません．しかし，限定的ではありますが，アプリケーションプログラムから呼び出す場合などある程度は使えるでしょうし，それらとリンクするときの基本的な指針としては十分だと思います．

本来は，デバイスドライバとして完成させたいところですが，それは読者への宿題としてバックカラーをサポートする画面システムとしてのプログラム作成はここでいったん締めたいと思います．

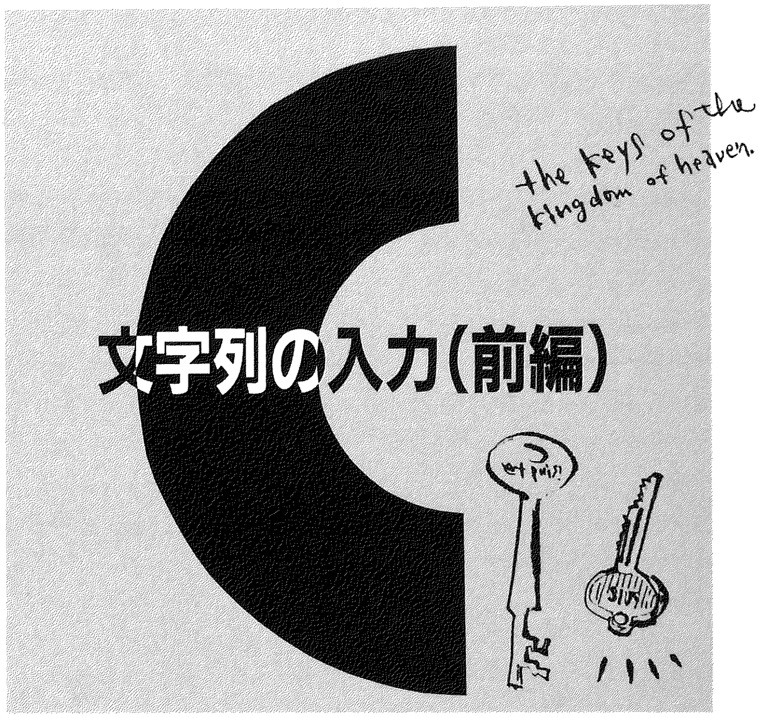
●参考文献

(1)共立出版　MS-DOS　纐纈一起・鷹野澄

Illustration : Masami Nozue

実践C塾

基礎テクニック篇　第2回



文字列の入力（前編）

二代目塾頭　大野元久

C 言語でプログラミングを行っていて，文字列の入力をしようとすると，標準関数の中に案外使える関数がないことに気がつく．
そこで，実践C塾では，今回と次回の2回にわたって特殊キーなどを正しく判別でき，文字列を編集できるといった特徴をもった文字列入力ルーチンを作成する．

プログラムを書いてみようというときに，意外に苦労するのが入力ルーチンです．たいていのCコンパイラには出力のための関数が数多く用意されているのですが，文字列を入力する関数で期待通りの動作をしてくれるものはほとんどありません．書式つき入力と呼ばれているscanfは，入力される文字を制限してくれないばかりか，ちょっとまちがえただけででたらめな動作を引き起こしたり，リターンを入力するだけでは抜けられないなど“誰もが知っている役に立たない関数”ともいえます．ちょっと工夫して1行入力のあとでsscanfで変換すれば若干使いやすくはなりますが(リスト1)，とても満足のいくものではありません．

文字列の入力で考慮しなければならないことは，たくさんあります．今回は，とくに重要だと思われる次のことに注意して汎用の文字列入力ルーチンを作成しました．

・［STOP］や［CTRL］+［C］を押してもプログラムが中断しないこと
・ファンクションキーや特殊キーを正しく判別できること
・カーソルキーで文字列を編集できること



[STOP], [CTRL]+[C]の扱い

Cの標準入力関数を使っている場合，［CTRL］+［C］（あるいは［STOP］）を押すと簡単にプログラムが終了してしまいます．コマンドラインコンパイラのように，ユーザーとの対話部分がほとんどないものではそのほうが望ましいこともありますが，多くのアプリケーションではこのような中断は望まれないものです．

リスト1 (lscanf.c)

```
/*
 *  lscanf.c: 1行単位での書式つき入力
 *       written by Motohisa Ohno
 *
 *  ※このプログラムは、Turbo C++ でのみ動作します。
 */
#include <stdio.h>
#include <stdarg.h>

int lscanf(char *fmt, ...)
{
     char buf[BUFSIZ];
     int n;
     va_list ap;

     if (gets(buf) == NULL)
         return 0;
     va_start(ap, fmt);
     n = vsscanf(buf, fmt, ap);
     va_end(ap);
     return n;
}

void main()
{
     int n, ret;

     do {
         printf("input number -> ");
         ret = lscanf("%d", &n);
         if (ret == 1)
             printf("number : %d¥n", n);
         else
             printf("illegal¥n");
     } while (n != 0);
}
```

DOSプログラムでは，次のような場合に［CTRL］+［C］の中断を受けつけます．

①［CTRL］+［C］による中断を受けつけるDOSの低レベルファンクション（01H～0CH）の呼び出し

低レベルDOSファンクション（01H～0CH）のうち，［CTRL］+［C］による中断を受けつけずに入出力ができるのはファンクション06H，07Hだけです．したがって，［CTRL］+［C］を受けつけないためには，これらのファンクションだけを使わなければなりません．

②ブレークチェックがオンになっているときのDOSファンクション（0DH～）の呼び出し

ブレークチェックがオンになっている場合は，0DH以降のほとんどのDOSファンクションを実行するたびに，［CTRL］+［C］が押されているかどうかを検査します．ブレークチェックとは，コマンドラインでの“BREAK”に相当します．たとえば，BREAK ONとすれば大きなファイルをコピーしているときでも［CTRL］+［C］で中断できることと同じです．通常，アプリケーションの中ではブレークチェックはオフにしておけばよいでしょう．

③COOKEDモードになっているキャラクタデバイスへのアクセス

通常，CON（コンソール入出力），AUX（補助入出力），PRN（標準プリンタ出力）などのキャラクタデバイスは，［CTRL］+［C］や［CTRL］+［S］などを受けつけるCOOKEDモードになっています．こうした特殊文字をそのまま受け渡しするモードをRAWモードといいます．これらのデバイスを使う場合は，デバイスをRAWモードにしなければなりません．しかし，標

準入出力については①のDOSファンクションを使うことができますし，補助入出力やプリンタ出力については標準付属のデバイスの機能が少なく，利用価値が低いと思われますので今回は考慮しません．

このモードを変更する場合は，リスト2のような関数を使ってください．ただし，デバイスをRAWモードにした状態でも②のブレークチェックがオンであれば［CTRL］+［C］による中断を受けます．

リスト2 (setdev.c)

```
#include <dos.h>

/*
 *  デバイスの制御文字の処理モードを変更
 *      入力    handle: ファイルハンドル
 *              mode: 1 (RAWモード), 0 (COOKEDモード)
 */
void setdev(int handle, int mode)
{
    union REGS regs;

    regs.h.ah = 0x44;
    regs.h.al = 0;
    regs.x.bx = handle;
    intdos(&regs, &regs);
    regs.h.ah = 0x44;
    regs.h.al = 1;
    regs.x.bx = handle;
    regs.h.dh = 0;
    regs.h.dl &= ~0x20;
    if (mode) regs.h.dl |= 0x20;
    intdos(&regs, &regs);
}

```

実際には，［CTRL］+［C］を押された場合に直ちに終了してしまうわけではなく，割り込みINT 23Hが発行され，割り込みハンドラの戻り値によって中断するかどうかが決まります．何も設定していなければ，デフォルトの動作，つまりプログラムの中断が行われるわけです．

では，上記の条件をいちいち回避するプログラミングをしなくても，中断割り込みのハンドラを設定しておけばプログラムを中断させずにすむのでしょうか？　そのとおりです．ただし，この割り込みハンドラが呼び出される前に必ず画面に“^C”という文字列が表示されてしまいます．これは，望ましいことではないでしょう．この文字列は標準出力に送られるため，この文字列が見たくなければ標準出力をクローズしてしまえばよいのですが，この場合，ほかの出力関数の動作に影響を与えることも考えられます．もっとも，多くのアプリケーションは標準出力を使わずに，直接VRAMにアクセスしていますから，この点での問題は少ないかもしれません．しかし，処理系によってはランタイムライブラリが割り込みハンドラを使って，浮動小数点エミュレータのベクタを初期化するために使ったりしますので，今回はこの割り込みについては処理しません．

PC-9801では，［STOP］が押されると割り込みINT 06Hが呼び出されます．これをトラップすれば，何かの処理を実行中に強制的に割り込むことができます．また，［STOP］を押すとハードディスクのリトラクトやカーソルの表示が行われますので，これらの処理を抑制したい場合にも，この割り込みをトラップしなければなりません．今回は，とくに必要がないためこの割り込みのトラップはしていません．



PC-9801では，割り込みINT DCHによる拡張ファンクションによって，ファンクションキーや特殊キーの内容を取得・設定します．

通常，ファンクションキーは単独でキーを押したものと［SHIFT］と組み合わせて使う20種類を使うことができます．［CTRL］+ファンクションキーという組み合わせは，ファンクションキーの表示を切り換えたり（［CTRL］+［f・7］），画面の行数を切り換えたり（［CTRL］+［f・6］）するためにMS-DOSが使用しています．最近のMS-DOSでは，この機能を無効にしてプログラムから［CTRL］+ファンクションキーという組み合わせを使えるようになっています．また，ハイレゾリューションモードの使える機種やRA2以降のものでは，拡張ファンクションキー（［vf・1］～［vf・5］）を使うことができます．

ファンクションキーには，各15文字までの文字列を設定できます．設定した文字列の先頭が文字コードFEHであれば，続く5バイトは表示のための仮文字列とみなされ，実際には7バイト目以降に設定された文字列が入力されます．

特殊キーとは，［ROLLUP］［ROLLDOWN］［INS］［DEL］，カーソルキー，［HOMECLR］（HOME），［HELP］，［SHIFT］+［HOMECLR］（CLR）を指します．よくまちがえられるのは，これらのキーをN88BASICの場合と同様に考えて，個別の文字コードが返されると考えてしまうことで

す．MS-DOSでは，これらのキーについて各５文字までの文字列を設定して，ファンクションキーと同様の使い方をします．何も設定していなければデフォルトで設定される文字コードが入力されますが，この文字コードをあてにして特殊キーを判別するプログラミングをしてはいけません．ユーザーは，MS-DOSに付属しているKEYコマンドを使って，これらの内容を書き換えているかもしれないからです．



これまでの内容を元にして，１文字入力ルーチンを作成します（リスト３，４，５）．

前回でも使用したMicrosoft CとTurbo C++との非互換性を吸収するturboc.hには，ブレークチェックに関するマクロが追加されています．また，Microsoft Cにはファンクションキーを扱う関数がないため，kbd.cの中でTurbo C++と同等の関数を定義しています．これらの関数については，プログラム中のコメントを参考にしてください．

各関数の仕様

■kbdinit

［機能］

入力ルーチンの初期化

［形式］

void kbdinit(void)；

［説明］

文字入力ルーチンを初期化します．すべての入力ルーチンを使用する前に，必ず一度だけ呼び出さなければなりません．

■kbdterm

［機能］

入力ルーチンの終了

［形式］

void kbdterm(void)；

［説明］

すべての文字入力ルーチンの使用を終了したら，アプリケーションを終了する前に，必ず一度だけ呼び出さなければなりません．

■setfkey

［機能］

ファンクションキーの表示文字列を設定

［形式］

void setfkey(int id, char *bufp)；

［説明］

idは１～20のファンクションキー番号（11～20は［SHIFT］＋ファンクションキー番号に相当する），bufpは５文字までの表示文字列を指すポインタです．初期状態では，すべてのファンクションキーには空文字列が設定されていますが，特定の機能をあらわすための文字列を設定するために使います．bufpにNULLを与えると，そのファンクションキーの表示文字列を空にします．setfkeyは，表示する文字列だけを変更するものでファンクションキーが返す文字コードを変更するわけではあ

リスト３（turboc.h）

```
/*
 *  Microsoft C のマクロ、構造体、関数の定義を Turbo C++ 形式に変更する。
 */
#ifndef __TURBOC_H
#define __TURBOC_H

#if !defined(__TURBOC__)

#if !defined(FP_SEG)
#include <dos.h>
#endif

/*
 *  ｆａｒポインタ用のマクロ
 */
#undef FP_SEG
#undef FP_OFF
#define FP_SEG(fp) ((unsigned)((unsigned long)(fp) >> 16))
#define FP_OFF(fp) ((unsigned)(fp))
#define MK_FP(seg,ofs) (((unsigned long)(unsigned)(seg)<<16) | (unsigned)(ofs))

/*
 *  ワイルドカード検索ルーチン
 */
struct ffblk {
    char ff_reserved[21];
    char ff_attrib;
    unsigned ff_ftime;
    unsigned ff_fdate;
    long ff_fsize;
    char ff_name[13];
};
#define findfirst(path,ffblk,attrib) ¥
    -(int)_dos_findfirst(path,attrib,(struct find_t *)ffblk)
#define findnext(ffblk) -(int)_dos_findnext((struct find_t *)ffblk)

#if !defined(_MAX_PATH)
#include <stdlib.h>
#endif

/*
 *  ファイル名操作ルーチン
 */
#define MAXPATH 80
#define MAXDRIVE 3
#define MAXDIR 66
#define MAXFILE 9
#define MAXEXT 5
#define fnsplit(path,drv,dir,file,ext) _splitpath(path, drv, dir, file, ext)
#define fnmerge(path,drv,dir,file,ext) _makepath(path, drv, dir, file, ext)

/*
 *  ブレークチェックの取得／設定
 */
#define getcbrk() (bdos(0x33, 0, 0))
#define setcbrk(cbrk) (bdos(0x33, (cbrk), 1))

#endif /* !defined(__TURBOC__) */

#endif  /* __TURBOC_H */

```

りません.

リスト4 (kbd.h)

```c
/*
 *  kbd.h: キー入力モジュールのための定義
 *      written by Motohisa Ohno
 */

/* ファンクションキーに設定する内部コードの1バイト目 */
#define FKEY_PRECODE 0x80
/* ファンクションキーに設定する内部コードの2バイト目の底上げ値 */
#define FKEY_BIAS 0x20

enum kbd_t {
    KBD_NULL = 0,
    /* CTRL+英字キー */
    KBD_CTRL_A, KBD_CTRL_B, KBD_CTRL_C, KBD_CTRL_D, KBD_CTRL_E, KBD_CTRL_F,
    KBD_CTRL_G, KBD_CTRL_H, KBD_CTRL_I, KBD_CTRL_J, KBD_CTRL_K, KBD_CTRL_L,
    KBD_CTRL_M, KBD_CTRL_N, KBD_CTRL_O, KBD_CTRL_P, KBD_CTRL_Q, KBD_CTRL_R,
    KBD_CTRL_S, KBD_CTRL_T, KBD_CTRL_U, KBD_CTRL_V, KBD_CTRL_W, KBD_CTRL_X,
    KBD_CTRL_Y, KBD_CTRL_Z,

    KBD_BS = 8, KBD_TAB = 9, KBD_CR = 13,   KBD_ESC = 27,
    KBD_CTRLMAX = 31,

    /* 特殊キーコードに対する底上げ値 */
    KBD_BIAS = 0x100,

    /* ファンクションキー */
    KBD_F1, KBD_F2, KBD_F3, KBD_F4, KBD_F5,
    KBD_F6, KBD_F7, KBD_F8, KBD_F9, KBD_F10,
    /* SHIFT+ファンクションキー */
    KBD_SF1, KBD_SF2, KBD_SF3, KBD_SF4, KBD_SF5,
    KBD_SF6, KBD_SF7, KBD_SF8, KBD_SF9, KBD_SF10,
    /* 特殊キー */
    KBD_RLUP, KBD_RLDN, KBD_INS, KBD_DEL,
    KBD_UP, KBD_LEFT, KBD_RIGHT, KBD_DOWN,
    KBD_HOME, KBD_HELP, KBD_CLR,

    KBD_LAST,
};

#if !defined(__TURBOC__)
/* MS-Cのためのファンクションキー取得・設定関数 */
void pc98getfkey(int id, void *bufp);
void pc98setfkey(int id, void *bufp);
void pc98fkeyon(void);
void pc98fkeyoff(void);
int pc98fkeychk(void);
#endif

void kbdinit(void);     /* 入力ルーチンの初期化 */
void kbdterm(void);     /* 入力ルーチンの終了 */
void setfkey(int id, char *bufp);   /* ファンクションキーの設定 */
void setfkeyall(char **buflist);    /* すべてのファンクションキーの設定 */
unsigned kbdback(unsigned c);       /* 1文字プッシュバック */
unsigned kbdchk(void);              /* 入力チェック */
unsigned kbdin(void);               /* 1文字入力（入力待ちあり） */
unsigned kbdscan(void);             /* 1文字入力（入力待ちなし） */
void kbdclr(void);                  /* 先行入力のクリア */
void setidlefunc(void (*fptr)(void));  /* 入力待ちで実行する関数の設定 */
```

■setfkeyall

[機能]

すべてのファンクションキーの表示文字列を設定

[形式]

void setfkeyall (char **buflist) ;

[説明]

最大20個までのファンクションキーに対する表示文字列を設定します. buflistは, 表示文字列をあらわすポインタ配列を指します. 配列にNULLがある場合は, それ以降の表示文字列を空に設定します. また, buflistにNULLを与えるとすべてのファンクションキーの表示文字列を空にします. setfkeyallは, 表示する文字列だけを変更するものでファンクションキーが返す文字コードを変更するわけではありません.

■kbdin

[機能]

1文字入力（入力待ちあり）

[形式]

unsigned kbdin(void) ;

[説明]

コンソールから1文字入力します. ファンクションキーや特殊キー, または漢字もそれぞれ1文字のコードとして返されます. ファンクションキーと特殊キーはkbd.hで定義されているKBD_xxというシンボルの値を返し, 漢字はシフトJISコード(0x8000以上)を返します. 1バイト目が漢字コードに該当し, 2バイト目が不正な漢字コードであればそのキー入力は無視され次の入力を待ちます. 1バイト文字は, 1～KBD_BIAS未満の値となります. 文字コード00H ([CTRL]+[@])を入力することはできません.

■kbdscan

[機能]

1文字入力（入力待ちなし）

[形式]

unsigned kbdscan(void) ;

[説明]

kbdinと同じですが, 入力待ちの文字がなければ直ちに戻り0を返します.

■kbdchk

[機能]

文字入力があるかどうかチェック

[形式]

unsigned kbdchk(void) ;

[説明]

入力待ちの文字があるかどうかをチェックします. 入力待ちの文字があればその文字コードを, なければ0を返します.

■kbdback

[機能]

1文字プッシュバック

[形式]

unsigned kbdback(unsigned c) ;

[説明]

キーバッファに1文字プッシュバックします. プッシュバックされた文字は, 次の文字入力関数の入力として使われます. すでにプッシュバックされている文字があるときは, エラーとして0が返されます. そうでなければプッシュバックした文字コードcを返します.

■kbdclr

[機能]

先行入力のクリア

[形式]

void kbdclr(void) ;

[説明]

先行入力して入力バッファにたまって

リスト5 (kbd.c)

```c
/*
 *  kbd.c: キー入力モジュール
 *      written by Motohisa Ohno
 */
#include <stddef.h>
#include <dos.h>
#include <jctype.h>
#include "turboc.h"
#include "kbd.h"

#if defined(__TURBOC__)
#include <pc98.h>
#else
/*
 *  ファンクションキーに関する関数（ＭＳ－Ｃ用）
 */

/*
 *  拡張ファンクションを使ってファンクションキーの内容を取得する
 *      入力:   id: ファンクションキー番号
 *            bufp: 取得用のバッファ
 */
void pc98getfkey(int id, void *bufp)
{
    union REGS regs;
    struct SREGS sregs;

    regs.h.cl = 0x0c;
    regs.x.ax = id;
    regs.x.dx = FP_OFF(bufp);
    sregs.ds = FP_SEG(bufp);
    int86x(0xdc, &regs, &regs, &sregs);
}

/*
 *  拡張ファンクションを使ってファンクションキーの内容を設定する
 *      入力:   id: ファンクションキー番号
 *            bufp: 設定される文字列バッファ
 */
void pc98setfkey(int id, void *bufp)
{
    union REGS regs;
    struct SREGS sregs;

    regs.h.cl = 0x0d;
    regs.x.ax = id;
    regs.x.dx = FP_OFF(bufp);
    sregs.ds = FP_SEG(bufp);
    int86x(0xdc, &regs, &regs, &sregs);
}

/*
 *  拡張ファンクションを使って文字列を出力する
 */
static void _dputs(char *str)
{
    union REGS regs;

    while (*str) {
        regs.h.cl = 0x10;
        regs.h.ah = 0;
        regs.h.dl = *str;
        int86(0xdc, &regs, &regs);
        ++str;
    }
}

/*
 *  ファンクションキーを表示する
 */
void pc98fkeyon(void)
{
    _dputs("¥033[>1l");
}

/*
 *  ファンクションキーの表示を消去する
 */
void pc98fkeyoff(void)
{
    _dputs("¥033[>1h");
}

/*
 *  ファンクションキーの表示状態をチェックする
 */
int pc98fkeychk(void)
{
    return *(unsigned char far *)MK_FP(0, 0x0711) != 0;
}

#endif


/* アプリケーション起動前のブレークチェックの状態 */
static int cbrk;
/* アプリケーション起動前のファンクションキーの内容 */
static char fkeybuf[386];

/*
 *  キー入力ルーチンの初期化
 */
void kbdinit(void)
{
    char buf[16];
    int i;

    cbrk = getcbrk();
    setcbrk(0);
    /* ファンクションキーの内容を退避する */
    pc98getfkey(0, fkeybuf);
    /* ファンクションキー名を空にして、内部コードを設定する */
    strcpy(buf, "¥xFE     ");
    buf[6] = FKEY_PRECODE;
    buf[8] = 0;
    for (i = 1; i <= 20; i++) {
        buf[7] = FKEY_BIAS + i;
        pc98setfkey(i, buf);
    }
    /* 特殊キーに内部コードを設定する */
    buf[0] = FKEY_PRECODE;
    buf[2] = 0;
    for (i = 21; i <= 31; i++) {
        buf[1] = FKEY_BIAS + i;
        pc98setfkey(i, buf);
    }
}

/*
 *  キー入力ルーチンの使用を終了する
 */
void kbdterm(void)
{
    pc98setfkey(0, fkeybuf);
    setcbrk(cbrk);
}

/*
 *  ファンクションキーの表示文字列を設定する
 *      入力:   id: ファンクションキー番号
 *            bufp: 設定される文字列（５文字まで）
 *                  bufp が NULL のときはクリア
 */
void setfkey(int id, char *bufp)
{
    char buf[16];

    if (id < 1 || id > 20)
        return;
    buf[0] = 0xFE;
    memset(buf + 1, ' ', 15);
    if (bufp) strncpy(buf + 1, bufp, 5);
    buf[6] = FKEY_PRECODE;
    buf[7] = FKEY_BIAS + id;
    buf[8] = 0;
    pc98setfkey(id, buf);
}

/*
 *  ２０個のファンクションキーを一度に設定する
 *      入力    buflist: ファンクションキーに設定する文字列リスト
 *                       リスト中にNULLがあればそれ以降はクリアされる
 *                       buflistがNULLのときはすべてクリアされる
 */
void setfkeyall(char **buflist)
{
    int i;

    i = 1;
    if (buflist) {
        while (buflist[i-1] && i <= 20) {
            setfkey(i, buflist[i-1]);
            ++i;
        }
    }
    while (i <= 20) {
        setfkey(i, NULL);
        ++i;
    }
}

/*
 *  文字入力ルーチンのための補助関数群
 *  入力は１バイト単位で処理され、他のモジュールから呼び出すことはできない。
 */

/*  １文字入力バッファ */
static int ch_sav = 0;

/*
 *  入力された文字があるかどうかをチェック
 */
static int _kbdchk(void)
{
    if (ch_sav == 0)
        ch_sav = (unsigned char)bdos(0x06, 0xFF, 0);
    return ch_sav;
}

/*
 *  １文字入力（入力待ちあり）
 */
static int _kbdin(void)
{
    int c;

    while (_kbdchk() == 0)
        ;
    c = ch_sav;
    ch_sav = 0;
    return c;
}

/*
 *  １文字入力（入力待ちなし）
 */
static int _kbdscan(void)
{
```

いる文字をキャンセルします．キー入力に対するプログラムの処理に時間がかかる場合，キーを押し続けたときの“慣性”動作を避けるためにkbdinなどの入力関数を呼び出す前にkbdclrを呼び出します．ただし，FEPを使って漢字を入力するときは漢字を確定したときに最初の1文字以外の入力がクリアされてしまいますので注意が必要です．

■setidlefunc

［機能］

入力待ちの間に実行される関数を設定

［形式］

void setidlefunc(void　(＊ fptr)(void))；

［説明］

入力待ちの間に実行される関数を設定します．ここで設定された関数は，kbdinを呼び出して実際に文字が入力されるまで繰り返し呼び出されます．画面の隅に時間を表示するような関数を設定しておけば，入力待ちの間は自動的に時間が更新されることになります．ただし，あまり関数の処理に時間がかかると，文字入力の“効き”が悪くなります．この場合は，何回かの呼び出しに1回ずつ処理を実行するような工夫が必要です．設定をキャンセルするときは，fptrにNULLを与えます．



ファンクションキー

入力ルーチンを初期化するkbdinitは，これらのキーが返す文字列として1バイト目を80H，2バイト目をキー番号（＋20H）に設定し，ブレークチェックをオフにします．kbdtermは，これらを初期化前の状態に戻します．

文字コード80Hは通常の文字入力では使われないため，入力された文字コードが80Hであればこれらの特殊キーであることが判別できます．2バイト目をキー番号そのものでなく20Hを加算しているのは，文字コード10H（[CTRL]＋[P]）が標準出力をプリンタにも送るかどうかを選択するために使われているなどの不都合が生じるためです．キー番号を表1に示します．

入力された文字コードが80Hであれば，次の文字コードが正しいキー番号をあらわしているかどうかを調べます．文字コードが正しければ，kbd.hで定義されているキー番号に相当する値（列挙型kbd_t）を返します．

```
    int c;

    c = _kbdchk();
    if (c) _kbdin();
    return c;
}

/*
 *  1文字入力ルーチン
 *  ファンクションキーや漢字も1文字として扱われる。
 *  漢字は、シフトJISであらわされる1ワードの値として返される。
 */

/* 1文字入力バッファ */
static unsigned ch_pushback = 0;

/*
 *  1文字プッシュバックルーチン
 *      入力    c: プッシュバックする文字コード
 *      出力    0: すでに文字がプッシュバックされている
 *              0以外: プッシュバックした文字コード
 */
unsigned kbdback(unsigned c)
{
    if (ch_pushback)
        return 0;
    return (ch_pushback = c);
}

/*
 *  入力があるかどうかのチェック
 *      出力    0: 入力待ちの文字はない
 *              0以外: 入力を待っている文字コード
 */
unsigned kbdchk(void)
{
    unsigned c, c2;

    if (ch_pushback == 0) {
retry:;
        c = _kbdscan();
        if (c == FKEY_PRECODE) {
            c = _kbdchk() - FKEY_BIAS + KBD_BIAS;
            ch_pushback = (KBD_F1 <= c && c <= KBD_LAST) ? c : 0;
            if (ch_pushback) _kbdin();
        } else if (iskanji(c)) {
            c2 = _kbdscan();
            if (!iskanji2(c2)) goto retry;
            ch_pushback = (c << 8) | c2;
        } else {
            ch_pushback = c;
        }
    }
    return ch_pushback;
}

/*
 *  入力待ちの間に実行される関数へのポインタ
 */
static void (*idlefunc)(void) = NULL;

/*
 *  1文字入力（入力待ちあり）
 *      出力    入力された文字コード
 */
unsigned kbdin(void)
{
    unsigned c;

    while (kbdchk() == 0)
        if (*idlefunc) (*idlefunc)();
    c = ch_pushback;
    ch_pushback = 0;
    return c;
}

/*
 *  1文字入力（入力待ちなし）
 *      出力    0: 入力を待っている文字はない
 *              0以外: 入力された文字コード
 */
unsigned kbdscan(void)
{
    unsigned c;

    c = kbdchk();
    if (c) kbdin();
    return c;
}

/*
 *  入力バッファのクリア
 */
void kbdclr(void)
{
    while (kbdscan())
        ;
}

/*
 *  入力待ちの間に実行される関数の設定
 *      入力    fptr: 入力待ちの間に実行される関数へのポインタ
 */
void setidlefunc(void (*fptr)(void))
{
    idlefunc = fptr;
}
```

入力された文字コードがシフトJISの1バイト目に相当する場合は，入力バッファを調べ正しいシフトJISの2バイト目が入力されていれば，これらを1ワードの漢字コードとして返します．めったにありませんが，1バイト目が入力された時点で2バイト目が入力バッファにない場合は1バイト目の入力は無視されます．PC-9801のMS-DOSでは，[CTRL]+[F5]を押したあと2桁の16進文字を入力して任意の文字コードを入力できますが，これを利用して漢字を入力することはできなくなります．もっとも，この機能はほとんど使われませんし，VJE-βなどが組み込まれている場合は正しく入力できますから問題はないでしょう．

これらの入力ルーチンでは，文字コード0（[CTRL]+[@]）は入力できないことに注意してください．文字コード0は，入力がないことやプッシュバックのエラーのために使われるため，キーボードから入力されても無視されてしまいます．また，次に作成する文字列入力ルーチンでは，文字列の最後に文字コード0を使用するため，これを通常文字と同等に扱うと不都合が生じてしまいます．

今回作成したルーチンは，「ファンクションキー，特殊キー」で述べた[vf･1]～[vf･5]，[CTRL]+ファンクションキーには対応していません．これは，すべての機種とMS-DOSでは使用できないためです．ソフトウェアを最近のDOSで使用することが明らかであれば，kbdinit，kbdtermを変更してこれらのキーにも対応することができるでしょう．ソフトウェアの中には[GRPH]+英字というキーの組み合わせを使うものもありますが，行儀のよい方法で対応することがなかなかむずかしいため，今回は対応していません．

入力ルーチンが文字入力のために使用しているDOSファンクションは，06Hのみです．これは，[CTRL]+[C]で中断されずに，入力待ちなしで文字コードを返す唯一の低レベルファンクションです．入力待ちを調べるDOSファンクション0BHは，[CTRL]+[C]によって中断されてしまうため，入力バッファはプログラムの中に用意しています．

すでに述べたように，標準入力デバイスをRAWモードにすれば[CTRL]+[C]による影響を受けずに標準入力から入力することもできるのですが，低レベルファンクションを使っているのは，致命的エラーハンドラから呼び出すこともできるという理由があります．致命的エラーハンドラから呼び出すことができるのは，01H～0CHの低レベルファンクションだけなので，それ以降のファンクションを使っていては都合が悪いのです．

文字入力のテストプログラムをリスト6に示します．kbdtest.cはkbd.cとともにコンパイル，リンクしてください．

このプログラムは，[f･1]～[f･5]と[INS]［DEL］キーを押すと対応するメッセージを表示し，漢字や半角文字を押すと文字コードを表示するもので

表1　キー番号一覧

キー番号	相当するキー
1～10	[f･1]～[f･10]
11～20	[SHIFT]+[f･1]～[SHIFT]+[f･10]
21	[ROLLUP]
22	[ROLLDOWN]
23	[INS]
24	[DEL]
25	[↑]
26	[←]
27	[→]
28	[↓]
29	[HOMECLR]（HOME）
30	[HELP]
31	[SHIFT]+[HOMECLR]（CLR）

図1

```
A>KBDTEST
任意のキーを押してください (ESC=終了)
[F1]ヘルプの表示
[F2]ブレークポイントの設定
1バイト文字: 文字コード=0x63
1バイト文字: 文字コード=0x68
1バイト文字: 文字コード=0x61
1バイト文字: 文字コード=0x72
[INS]挿入
[DEL]削除
漢字: 文字コード=0x8abf[漢]
漢字: 文字コード=0x8e9a[字]
1バイト文字: 文字コード=0x1b
```

リスト6（kbdtest.c）

```c
/*
 *  kbdtest.c: キー入力ルーチンのテスト
 *      written by Motohisa Ohno
 */
#include <stdio.h>
#include "kbd.h"

/* ファンクションキー表示のための文字列 */
char *fkeylist[] = {
    "Help", "Bkpt", "Mod", "Here", "Zoom", NULL
};

void main()
{
    unsigned code;

    kbdinit();
    setfkeyall(fkeylist);
    printf("任意のキーを押してください (ESC=終了)¥n");
    do {
        code = kbdin();
        switch (code) {
        case KBD_F1: puts("[F1]ヘルプの表示"); break;
        case KBD_F2: puts("[F2]ブレークポイントの設定"); break;
        case KBD_F3: puts("[F3]モジュールのオープン"); break;
        case KBD_F4: puts("[F4]カーソル行まで実行"); break;
        case KBD_F5: puts("[F5]ズーム/アンズーム"); break;
        case KBD_INS: puts("[INS]挿入"); break;
        case KBD_DEL: puts("[DEL]削除"); break;
        default:
            if (code > 0x8000) {    /* 漢字 */
                printf("漢字: 文字コード=%#04x[%c%c]¥n",
                    code, (code >> 8), (unsigned char)code);
            } else if (0 <= code && code < KBD_BIAS) {  /* 半角文字 */
                printf("1バイト文字: 文字コード=%#02x¥n", code);
            }
            break;
        }
    } while (code != KBD_ESC);  /* ESC で終了 */
    kbdterm();
}

```

す．入力ルーチンを使うプログラムの先頭と最後では，それぞれkbdinitとkbdtermを呼び出さなければなりません．これを忘れると，ファンクションキーや特殊キーが正しく認識されなかったり内容が書き換えられたままプログラムを終了することになります．

プログラムの中心となるのは，do { code = kbdin () ; . . . } while (code != KBD_ESC) ; の繰り返し文です．kbdinで入力された文字コードをswitchブロックのcase文で分類しています．プログラムを終了させるときは，[ESC] を押してください．

文字列入力ルーチン

次に文字列を入力するための関数を作成します（リスト7，8，9，10）．

tvramは，文字列入力ルーチンのために作成したテキスト画面をアクセス

リスト7 (editstr.h)

```
/*
 *  editstr.h: 文字列入力モジュールのための定義
 *      written by Motohisa Ohno
 */

typedef struct {
    int lx, ly:         /* 編集位置の開始座標：左上が（０，０） */
    int width;          /* 編集位置の幅 */
    int length;         /* 文字バッファの最大長（ヌル文字を含む） */
    int offset;         /* 表示位置オフセット（先頭＝０） */
    int curp;           /* 現在のカーソル位置（先頭＝０） */
    int first;          /* 編集の最初かどうかのフラグ（最初＝１） */
    int type;           /* 編集文字列の種類 */
    char *bufp;         /* 文字列バッファ */
    int histn;          /* ヒストリー番号 */
    int histlen;        /* ヒストリーバッファの長さ */
    char *histp;        /* ヒストリーバッファへのポインタ */
} EDITSTR;

/* 編集状態 */
enum editstr_t {
    /* 文字列編集の最初 */
    EDS_BEGIN = 0x0800,

    /* 編集領域の左に文字がある */
    EDS_LEFT = 0x1000,
    /* 編集領域の右に文字がある */
    EDS_RIGHT = 0x2000,
    /* 挿入モードがオン */
    EDS_INSERT = 0x4000,
};

enum edittype_t {
    /* 編集する文字の種類 */
    EDT_DIGIT = 0x0001,
    EDT_ALPHA = 0x0002,
    EDT_KANA = 0x0004,
    EDT_HANKAKU = 0x0008,
    EDT_KANJI = 0x0010,

    /* ＦＥＰの制御 */
    EDT_USEFEP = 0x1000,
    EDT_NOUSEFEP = 0x2000,
};

extern int insert_mode; /* 挿入モード */

unsigned editstr(unsigned code, EDITSTR *ep);    /* 文字列入力 */

```

リスト8 (editstr.c)

```
/*
 *  editstr.c: 文字列入力モジュール
 *      written by Motohisa Ohno
 */
#include <string.h>
#include <jctype.h>
#include <jstring.h>
#include <dos.h>
#include "turboc.h"
#include "kbd.h"
#include "tvram.h"
#include "editstr.h"

int insert_mode = 1;                    /* 挿入モード */
unsigned edit_attr = 0x00E1;            /* 入力文字の属性 */

/*
 *  文字列入力ルーチン
 *      入力    code: 入力された文字コードあるいはEDS_BEGIN
 *              ep: 文字列入力構造体（EDITSTR）へのポインタ
 *      出力    文字列入力の状態
 */
unsigned editstr(unsigned code, EDITSTR *ep)
{
    int len;
    int i, n, m;

    len = strlen(ep->bufp);
    switch (code) {
    case EDS_BEGIN:
        /* 文字入力を行なうための初期化処理 */
        if (len < ep->width) {
            ep->offset = 0;
        } else {
            ep->offset = len - ep->width + 1;
        }
        ep->curp = len;
        break;

    case KBD_RIGHT:
        /* カーソルを右へ移動する */
        if (ep->curp < len) {
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            ep->curp += n;
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            if (ep->width < (ep->curp - ep->offset + n))
                ep->offset = ep->curp - ep->width + n;
        }
        break;

    case KBD_BS:
        /* バックスペース　＝　左へ移動＋文字の削除 */

    case KBD_LEFT:
        /* カーソルを左へ移動する */
        if (ep->curp > 0) {
            ep->curp--;
            if (nthctype(ep->bufp, ep->curp) == CT_KJ2)
                ep->curp--;
            if (ep->offset > ep->curp)
                ep->offset = ep->curp;
        }
        if (code == KBD_LEFT) break;

    case KBD_DEL:
        /* カーソル位置の文字を削除 */
        if (ep->curp < len) {
            if (iskanji((unsigned char)(ep->bufp[ep->curp])))
                n = 2;
            else
                n = 1;
            for (i = ep->curp + n; i < len; i++)
                ep->bufp[i-n] = ep->bufp[i];
            ep->bufp[len-n] = 0;
        }
        break;

    case KBD_INS:
        /* 挿入モードの切り換え */
        insert_mode = !insert_mode;
        break;

    case KBD_HOME:
        /* 文字列の先頭へ移動 */
        ep->curp = 0;
        ep->offset = 0;
        break;

    case KBD_CLR:
        /* 文字列の最後へ移動 */
        if (len < ep->width) {
            ep->offset = 0;
        } else {
            ep->offset = len - ep->width + 1;
        }
        ep->curp = len;
        break;

    case KBD_NULL:
    case KBD_CR:
    case KBD_ESC:
        /* 入力文字列としては使わない */
        break;

```

```c
        default:
            /* リターン、ESC以外の通常文字は、すべて入力の対象となる */
            if (code == 0 || (KBD_BIAS <= code && code < 0x8000))
                break;
            /* 初期化直後に文字が入力されたときは */
            /* デフォルト文字列をクリアしてから入力する */
            if (ep->first) {
                ep->curp = 0;
                ep->offset = 0;
                ep->bufp[0] = 0;
                len = 0;
            }
            n = (code < 0x8000) ? 1 : 2;
            m = (insert_mode ? 0 : (iskanji(ep->bufp[ep->curp]) ? 2 : 1));
            if ((len + n - m) >= ep->length)
                break;
            if (m < n) {
                for (i = len; i >= ep->curp; i--)
                    ep->bufp[i+n-m] = ep->bufp[i];
            } else if (m > n) {
                for (i = ep->curp-n+m; i <= len; i++)
                    ep->bufp[i+n-m] = ep->bufp[i];
            }
            if (n > 1) ep->bufp[ep->curp++] = (code >> 8);
            ep->bufp[ep->curp++] = (unsigned char)code;
            if (ep->width <= (ep->curp - ep->offset))
                ep->offset = ep->curp - ep->width + 1;
            break;
        }
        ep->first = (code == EDS_BEGIN);
        /* 編集文字列の表示 */
        if (nthctype(ep->bufp, ep->offset) == CT_KJ2) {
            dspchr(ep->lx, ep->ly, ' ', edit_attr);
            dspstr(ep->lx+1, ep->ly, ep->width-1, ep->bufp+ep->offset+1, edit_attr);
        } else {
            dspstr(ep->lx, ep->ly, ep->width, ep->bufp + ep->offset, edit_attr);
        }
        /* カーソル位置の設定 */
        setcurpos(ep->lx + ep->curp - ep->offset, ep->ly);
        /* 編集状態の設定 */
        len = strlen(ep->bufp);
        n = 0;
        if (ep->offset) n |= EDS_LEFT;
        if ((ep->offset + ep->width) < len) n |= EDS_RIGHT;
        return n;
}
```

するモジュールです．Turbo C++にもMicrosoft Cにもテキスト画面にアクセスするライブラリ関数はあるのですが，残念ながら両者の間にはほとんど互換性がありません．このため独自のモジュールを作成しましたが，ライブラリ関数はアセンブラで記述されていることも多いためこれらの関数をライブラリのテキスト処理関数を使って書きなおすと効率がよくなるかもしれません．

リスト 9 （tvram.h）

```c
/*
 *  tvram.h: テキストVRAMモジュールのための定義
 *      written by Motohisa Ohno
 */

void tvraminit(void);
void tvramterm(void);
int dspchr(int lx, int ly, unsigned code, unsigned attr);
int dspstr(int lx, int ly, int len, char *bufp, unsigned attr);
void setcurpos(int lx, int ly);
```

今回作成した文字列入力ルーチンのためには，基本的にdspchr，dspstr，setcurposの３つの関数だけがあればよいのですが，ハイレゾリューションモードにも対応できるように，テキストVRAMのセグメントを検出するための初期化関数（tvraminit）を用意しています．

リスト 10 （tvram.c）

```c
/*
 *  tvram.h: テキストVRAMモジュール
 *      written by Motohisa Ohno
 */

#include <stdio.h>
#include <stdlib.h>
#include <jctype.h>
#include <jstring.h>
#include <dos.h>
#include "turboc.h"
#include "tvram.h"

static unsigned (far *tvram)[80] = (unsigned (far *)[80])0xA0000000L;
static unsigned (far *avram)[80] = (unsigned (far *)[80])0xA0002000L;

/*
 *  テキストVRAMモジュールのための初期化
 */
void tvraminit(void)
{
    unsigned tseg;

    tseg = (*(unsigned char far *)MK_FP(0x0000, 0x0501) & 0x08)
                ? 0xE000 : 0xA000;
    tvram = (unsigned (far *)[80])MK_FP(tseg, 0);
    avram = (unsigned (far *)[80])MK_FP(tseg, 0x2000);
}

/*
 *  テキストVRAMモジュールの終了
 */
void tvramterm(void)
{
}

/*
 *  1文字出力
 */
int dspchr(int lx, int ly, unsigned code, unsigned attr)
{
    static unsigned kflag = 0;

    if (code == 0) {
        if (kflag == 0) return 0;
        kflag = 0;
        code = ' ';
    }
    code = (unsigned char)code; /* XXX */
    if (kflag) {
        code = (kflag << 8) | code;
        code = jmstojis(code) - 0x2000;
        code = (code >> 8) | (code << 8);
        tvram[ly][lx] = code;
        avram[ly][lx] = attr;
        tvram[ly][lx+1] = code;
        avram[ly][lx+1] = attr;
        kflag = 0;
        return 2;
    }
    if (iskanji(code)) {
        kflag = code;
        return 0;
    }
    tvram[ly][lx] = code;
    avram[ly][lx] = attr;
    return 1;
}

/*
 *  文字列出力
 */
int dspstr(int lx, int ly, int len, char *bufp, unsigned attr)
{
    int lxbgn = lx;

    while (len && *bufp) {
        lx += dspchr(lx, ly, (unsigned char)*bufp++, attr);
        --len;
    }
    while (len) {
        lx += dspchr(lx, ly, ' ', attr);
        --len;
    }
    lx += dspchr(lx, ly, 0, attr);
    return lx - lxbgn;      /* must be equal to 'len' */
}

/*
 *  カーソル位置の設定
 */
void setcurpos(int lx, int ly)
{
    union REGS regs;

    regs.h.ah = 0x13;
    regs.x.dx = ly * 160 + lx * 2;
    int86(0x18, &regs, &regs);
}
```


ワンラインNEWS
YHPは業界最小・最軽量のパームトップコンピュータ「HP 95LX」を発売した．ロータス社の表計算ソフト「1-2-3 R2.2」を搭載しているのが特徴

文字列入力ルーチンといっても，ここで作成したeditstrは単独では文字列を入力することができません．多くの場合，文字列の入力中にファンクションキーや何かのキーが押されたら，別の処理に移行したいことがあるものです．そこで，文字入力はeditstrの外で行い，editstrには必要な文字だけが渡されることにしました．このため，文字列入力は文字入力ルーチンとeditstrの呼び出しの繰り返しになります．

editstrを使ったもっとも簡単な文字列入力ルーチンは，リスト11のようになります．inputstr.cは，editstr.c，tvram.c，kbd.cとともにコンパイル，リンクしてください．

このテストプログラムを使って，editstrの使い方を説明しましょう．

まず，プログラムの先頭と最後で，入力とテキスト画面の初期化と終了のための関数を呼び出すことを忘れないでください．これらを呼び出さなければ，正常な文字列入力はできません．

文字列入力を行うためには，必ず構造体EDITSTR型の変数を定義しなければなりません．この構造体は，editstr.h中で定義され，文字列入力を行う画面上の座標，幅，文字列を格納するバッファへのポインタやバッファの長さなどが含まれています．メンバtype，histlen，histpは，今の段階では使いません．このほかのメンバは，editstrの内部で使用されます．文字列を入力する前に，EDS_BEGINとともに各メンバを設定した構造体へのポインタをeditstrに渡します（editstr(EDS_BEGIN，ep);）．文字列バッファには，あらかじめ編集される文字列を格納しておかなければならないことに注意してください．空文字列を渡すときは，文字列バッファの先頭に文字コード0を入れておきます．editstrは，渡された文字列を入力文字列のデフォルトとして使用します．

初期設定が終わったら，あとはkbdinとeditstrを交互に呼び出します．文字列を入力する目的以外で使用する文字コードがあれば，editstrに渡す前に処理します．ここでは，KBD_CR(リターン)で入力の終了を判定しています．基本的に，kbdinで入力された文字コードはすべてeditstrに渡してかまいません．editstrは，カーソルキーなどの文字の編集に使われるものを判別しますし，編集に必要のない文字コードは単に無視します．漢字や半角文字はすべて文字入力とみなしますので，数値だけとか漢字だけを入力したいというように入力文字を制限する場合は，入力したくない文字をeditstrに渡さなければよいのです．また，KBD_NULLをeditstrに渡しても入力は無視されます．

editstrに渡す文字コードは，kbdinから渡されたものそのものでなくてもかまいません．たとえば，[CTRL]+[S]や[CTRL]+[D]が入力されたときに，これらをKBD_LEFTやKBD_RIGHTの文字コードに変換して渡せば，これらのキーをカーソルを移動させるために使えます．

もう少し複雑な使用例をリスト12に示します．これも，editstr.c，tvram.c，kbd.cとともにコンパイル，リンクします．

getstrでは，editstrからの戻り値を利用しています．editstrは，入力のために画面で使用する桁幅よりも文字列が長い場合は，文字列を左右にスクロールしますが，左右に文字列がスクロールアウトしているかを示すビットを返します．getstrは，これらを利用して

リスト11 (inputstr.c)

```
/*
 *  inputstr.c: editstr関数の使用例
 *       written by Motohisa Ohno
 */
#include <stdio.h>
#include <stdlib.h>
#include "kbd.h"
#include "editstr.h"

/*
 *  文字列入力関数
 */
void inputstr(EDITSTR *ep)
{
    unsigned code;

    editstr(EDS_BEGIN, ep);
    do {
        code = kbdin();
        editstr(code, ep);
    } while (code != KBD_CR);
}

void main()
{
    EDITSTR es;
    char buf[20];

    tvraminit();
    kbdinit();

    es.lx = 5;
    es.ly = 3;
    es.width = es.length = 20;
    buf[0] = 0;
    es.bufp = buf;

    printf("¥033[2J");  /* 画面の消去 */
    inputstr(&es);
    printf("input: [%s]¥n", es.bufp);

    kbdterm();
    tvramterm();
}
```

スクロールアウトしているかどうかの印を表示します．文字列の入力は，ヒープメモリから確保した一時バッファを利用し，リターンが入力された時点ではじめて渡された文字列バッファにコピーされます．ESCが入力されると渡された文字バッファの内容を変更せずに戻ります．

[CTRL]+[S] や [CTRL]+[D] は，適当なほかの文字コードを変換して，ダイヤモンドカーソルのような操作を行っています．それ以外の制御文字は無効な入力に変換されます．

テストプログラムでは，“DEFAULT”という文字列をデフォルトの文字列として与えています．デフォルトの文字列が表示された直後に文字を入力すると，デフォルトの文字列は消去され，その文字列が入力バッファの先頭の文字として使われます．デフォルトの文字列を編集する場合は，カーソルキーなどほかの特殊キーを押してから行います．

editstrのプログラミングについては，漢字を扱うためにあまりきれいなコードでなくなっていることもありますので，実際のソースコードを参照してください．もし，別のキー入力に対して編集機能を追加する場合は，switch文に新しいcase文を追加して，文字バッファの内容や構造体の各メンバを変更するようにプログラミングします．文字バッファの表示や編集状態の設定は，switch文のあとで行っていますので必要ありません．

来月は，この関数にヒストリー機能を組み込んでみたいと思います．お楽しみに．

リスト12　(estest.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "kbd.h"
#include "tvram.h"
#include "editstr.h"

unsigned edit_mark_attr = 0x00A1;   /* ←→マークのための文字属性 */

int getstr(EDITSTR *ep)
{
    char *p;
    unsigned code, state;

    /* ヒープメモリから編集用の一時バッファを確保 */
    p = strdup(ep->bufp);
    if (p == NULL) return 0;
    strcpy(p, ep->bufp);

    /* 文字列入力の初期化 */
    state = editstr(EDS_BEGIN, ep);
    do {
        dspchr(ep->lx-1, ep->ly,
                (state & EDS_LEFT) ? 0x1D : ' ', edit_mark_attr);
        dspchr(ep->lx+ep->width, ep->ly,
                (state & EDS_RIGHT) ? 0x1C : ' ', edit_mark_attr);
        code = kbdin();
        switch (code) {
        case KBD_CR:
            /* リターンが入力されたら文字列設定 */
            break;
        case KBD_ESC:
            /* ＥＳＣのときは文字列を元に戻す */
            strcpy(ep->bufp, p);
            break;
        default:
            /* 文字コードの変換 */
            switch (code) {
            case KBD_CTRL_S: code = KBD_LEFT; break;
            case KBD_CTRL_D: code = KBD_RIGHT; break;
            case KBD_CTRL_A: code = KBD_HOME; break;
            case KBD_CTRL_F: code = KBD_CLR; break;
            case KBD_CTRL_G: code = KBD_DEL; break;
            case KBD_BS: break;
            default:
                if (0 <= code && code <= KBD_CTRLMAX)
                    code = KBD_NULL;
                break;
            }
            state = editstr(code, ep);
            break;
        }
    } while (code != KBD_CR && code != KBD_ESC);
    /* 一時バッファの解放 */
    free(p);
    return code;
}

void main()
{
    char buf[30];
    EDITSTR es;

    tvraminit();
    kbdinit();
    printf("¥033[2J");  /* 画面の消去 */

    es.lx = 10;
    es.ly = 5;
    es.width = 20;
    es.length = 30;
    es.bufp = buf;
    strcpy(es.bufp, "DEFAULT");

    getstr(&es);
    printf("your input is [%s]¥n", es.bufp);

    kbdterm();
    tvramterm();
}

```


ワンラインNEWS
日本IBMはオフコン「AS/400」の全モデルを一新し，「AS/400・Dモデル」として出荷開始した．最上位機「D80」は従来モデルの2.75倍の処理能力をもつ．

情報処理技術者試験
第2種合格！トレーニング
第5回　工　業

日本工学院専門学校　小宮正好

■傾向と対策

工業系の問題は午前選択問題10問のうちから毎年2問ないし4問が出題されています．過去の出題をまとめると**表1**のようになります．

一般常識の問題は高校在学中の受験生を対象とした問題であると考えてよいでしょう．この問題は普通科高校の物理の問題や工業高校の機械製図の問題などが出題されていますが，同じ問題は出題されていません．同様の傾向は「その他の分野」についてもあてはまります．このために受験者の中には，「工業系は出題範囲が広く対策の立てようがない」という印象をもつ人もいるようです．

たしかに一般常識に関しては高校の勉強をまじめに行う以外にこれといった対策は立てられないでしょう．

しかしながら，品質管理から待ち行列までの分野は過去の試験において繰り返し出題されていることが表1からわかります．これらの分野を勉強しておくことは効率的な受験勉強ということができるでしょう．本講座ではページ数の関係から今後数回の試験でとくに出題確率が高いと考えられる品質管理と在庫管理の概略を説明します．

なお，各分野からの出題は基本的な問題が大部分を占めています．各分野の出題ポイントを**表2**にまとめておきましたので勉強の参考にしてください．

●表1　過去の出題範囲

単元＼年度	出題回数	昭和50年	昭和51年	昭和52年	昭和53年	昭和54年	昭和55年	昭和56年	昭和57年	昭和58年	昭和59年	昭和60年	昭和61年春	昭和61年秋	昭和62年春	昭和62年秋	昭和63年春	昭和63年秋	平成元年春	平成元年秋	平成2年春	平成2年秋	平成3年春
品質管理	8		16						15		15		16	17	16		16					16	
在庫管理	8		15		16	17	16				16							16		18	14		
生産管理	8	18		20									18		17	17	18					18	17
日程計画法	5				14		15												18		16		16
計測と制御	3									15			17	18									
線形計画法	2							13									17						
待ち行列	2	17																		17			
信頼性工学	1							14															
一般常識	7											17		16	18	16		18	16		17		
その他	8					18		18				16							17	16	18	17	18

マス目の数字は問題番号を示す

●表2　各分野の出題のポイントとその対策

■在庫管理	在庫の特性を決定するABC分析について理解しておく． 在庫の特性を考慮して合理的に設計された管理方式について理解しておく（定期発注方式，発注点方式）． 発注点方式については，最適発注量を導き出せるようにしておく．
■品質管理	品質管理の意味とその基本概念（品質特性と管理限界など）について理解しておく． 管理図の理論的基礎となる検定の概念と用語（帰無仮説，対立仮説，有意水準，危険率，第1種の誤り，第2種の誤り，あわて者の誤り，のんびり者の誤り）について理解しておく． 管理図の種類と適用対象について理解しておく． 抜き取り検査の種類と適用対象について理解しておく．
■生産管理	組立ラインに関する計算に習熟しておく． 歩留に関する計算に習熟しておく． 部品展開について理解しておく． 部品量の計算に習熟しておく． MRPについて理解しておく．
■日程計画	PERTの基本概念を理解し，基本的な値（最早（最遅）結合点時刻，最早（最遅）開始（終了）時刻，全余裕，自由余裕）が計算できるようにしておく．
■計測と制御	基本的な計測機器について学習しておく． 基本的な制御方式の概要を理解しておく．
■線形計画法	与えられた文章を線形改革問題として定式化できるようにしておく． 線形計画問題の図形的解法を理解しておく．
■待ち行列	待ち行列の基本的構造を理解しておく． 基本的な公式を理解し使えるようにしておく．

（合格情報処理，1991年1月号，㈱学習研究社，「ラジオたんぱ　第2種情報処理講座」，小宮正好担当）

■品質管理

EXAMPLE 1

品質管理で用いる統計的方法に関する次の記述の中から正しいものを5つ選べ．

ア　“差がない”，“効果がない”というような形の仮説を帰無仮説という．

イ　帰無仮説と対立する仮説を対立仮説という．

ウ　測定値から計算した差異が，帰無仮説を捨てるに足るほど大きいことを有意という.

エ　帰無仮説が真でないにもかかわらず帰無仮説を捨てない誤りを第1種の誤りという.

オ　第2種の誤りの確率を危険率という.

カ　危険率は検定に先立ち目的に応じてあらかじめ定めておくことが必要である.

キ　1から第2種の誤りの確率を引いたものを検出力という.

ク　第1種の誤りをぼんやりものの誤りともいう.

ケ　第1種の誤りの確率を消費者危険ともいう.

コ　平均値からの差を不偏分散で割ったものが偏差値である.

昭和62年　春期　第2種　問17

解　説

管理図の理論的基礎となる検定についての出題です.

製造工程が正常ならば製品の重量の分布が判っている場合に検定の考え方を適用してみましょう．検定の考え方を適用することによりサンプルとして取り出した1つの製品の重量を調べることで製造工程の正常，異常に関する判定を合理的に行うことができます.

具体的には工程が正常であるという仮説を立ててこの仮説が正しいかどうかを検討することになります.

まず，仮説が成立していたときに重量の大部分が含まれる範囲を管理限界として設定します．管理限界の範囲としては，通常99%，もしくは95%の値が含まれる範囲が設定されます(**図1**).

次に，実際に1つの値を測定してみます．この値が管理限界線から外れていたとしましょう．仮説が正しければこのデータの得られる確率は低いわけです．したがって，この時ははじめに立てた仮説が誤っていると判断することになります．これに対して管理限界内の値が測定されたときは仮説が正しいと判断することになります.

これが検定の考え方です．ここまで読まれた方はこの考え方が合理的であると感じられたのではないでしょうか．事実この考え方は数学的合理性に裏づけされています．次に，いままで述べたことを整理してこの点を明らかにしてみましょう.

まず，はじめに立てた仮説は正しいか誤っているかのどちらかであるはずです．これに対して，下される判断は仮説を正しいとする判断か誤っているとする判断かのどちらかになります.

したがって次の2つの場合に誤った判断が下されることになります.

1．正しい仮説を誤っていると判断する．この誤りをあわてものの誤り，もしくは第1種の誤りといいます.
2．誤っている仮説を正しいと判断する．この誤りをのんびりものの誤り，もしくは第2種の誤りといいます.

第1種の誤りを犯す確率は値が管理限界線から外れる確率に等しく，5%，もしくは1%ということになります．これに対して第2種の誤りを犯す確率は仮定が成立しないときの分布の形が不明なので計算することはできません.

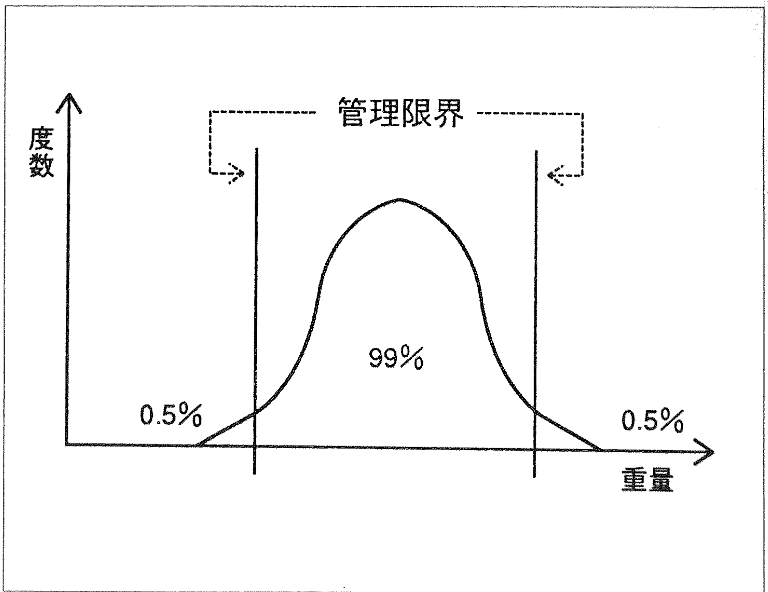
したがって，はじめに立てた仮説を否定する判断はその判断が誤っている確率を計算できるという意味で有用な判断であると考えられます．これから，はじめの仮説は否定されることを期待して立てる仮説という意味で帰無仮説(注：無に帰することを期待して立てる仮説という意味)と呼ばれます．これに対して帰無仮説に対立する仮説は対立仮説と呼ばれています．この用語を使えば帰無仮説が成立しないと判断することは対立仮説が成立すると判断することになります.

以上のことは**図2**のように整理されます.

解　答

ア，イ，ウ，カ，キ

●図1



●図 2

	帰無仮説が成立	対立仮説が成立
帰無仮説が成立していると判断する	正しい判断	のんびり者の誤り
対立仮説が成立していると判断する	あわて者の誤り	正しい判断

EXERCISE 1

管理図に関する次の記述中の［ ］に入れるべき最も適切な字句を，解答群の中から選べ．解答は重複して選んでもよい．

管理図は，品質や製造条件などの時間的変化を図にプロットして工程を管理しようとするものであり，判断の誤りに対して客観的な基準が用いられる．

3シグマ法の$\bar{x}$管理図では，［ a ］をおかす確率が0.3%となるように管理限界線が設定される．製造工程を管理する場合，［ b ］から計算した［ c ］$\bar{x}$を記入し，この点が管理限界線の外側にあるか内側にあるかで管理状態を判断するのが基本である．点がちょうど限界線上のときは，［ d ］とみなすことになっている．

管理限界線を広くすれば，［ a ］をおかす確率は小さくなるが，［ e ］をおかす確率が大きくなる．また，3シグマ法の管理限界を1.96シグマにすると，［ f ］をおかす確率は小さくなるが，［ a ］をおかす確率が0.3%から［ g ］となってしまう．

これらの誤りを全くおかさないようにすることは［ h ］である．

一般に$\bar{x}$管理図は［ i ］と併用して用いられ，工程に［ j ］が生じていることを早期に感知することを目的とする．

〈a,e,fに対する解答群〉

ア．第１種の誤り
イ．第２種の誤り
ウ．第３種の誤り　エ．計算間違い
オ．情報伝達間違い カ．指示間違い

〈b,jに対する解答群〉

ア．管理状態　　イ．試料
ウ．母集団　　エ．偶発変動
オ．異常変動

〈c,iに対する解答群〉

ア．期待値　　イ．平均値
ウ．偏差値　　エ．範囲
オ．Z管理図　　カ．p管理図
キ．R管理図　　ク．n管理図

〈d,hに対する解答群〉

ア．可能　イ．不可能　ウ．一般的
エ．例外的　オ．限界内　カ．限界外
キ．経済的　ク．不経済

〈gに対する解答群〉

ア．0.03%　イ．0.1%　ウ．1%
エ．3%　オ．5%

平成２年度　秋期　問16

※ポイント

管理限界を横向きに書き込み検定を連続して行えるようにした図は管理図と呼ばれています．

解　答

a－ア，b－イ，c－イ，d－オ，
e－イ，f－イ，g－オ，h－イ，
i－キ，j－オ

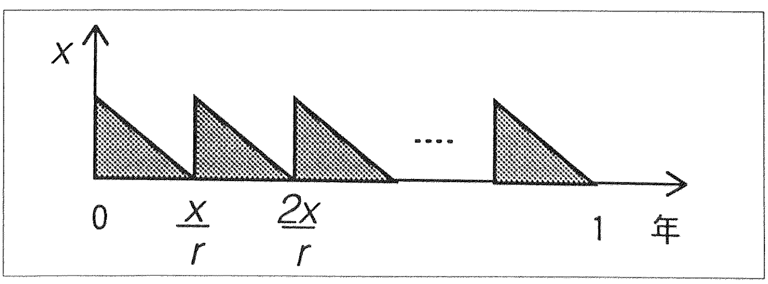
■在庫管理

EXAMPLE 2

在庫管理に関する次の記述中の［ ］に入れるべき適切な字句を，解答群の中から選べ．

需要が一定で品切れは発生しない状況での，ある商品の在庫管理について考える．発注した商品は即時に納入され，毎回の発注量は一定とする．このとき，図のように納入された商品は毎日一定量ずつ減少し，在庫がなくなったとき一定の発注量が即時納入される．1回の発注量をx個，1回あたりの発注量をc_0円，商品1個あたりの在庫保管費（1年間）をc_1円，年間の総需要をr個とする．商品の個数を連続量とみるとき，年間の発注回数は［ a ］回，年間の平均在庫量は，図の三角形の部分の面積の和であるから［ b ］個となる．年間の総費用Cを年間の発

●在庫量



注費と年間の在庫保管費の和とすると，C は次のような x の関数となる．

$C=$［ c ］

C を x で微分して，C を最小にする1回の発注量を求めると［ d ］個，そのときの発注間隔（発注から次の発注までの時間）は，［ e ］年，総費用 C の最小値は［ f ］円となる．

〈a，bに関する解答群〉

ア．$\frac{x}{2}$　イ．$\frac{x^2}{r}$　ウ．$\frac{x^2}{2r}$

エ．$\frac{x}{r}$　オ．$\frac{r}{x}$

〈cに関する解答群〉

ア．$\frac{C_0x}{r}+\frac{C_1x^2}{r}$　イ．$\frac{C_0r}{x}+\frac{C_1x^2}{2r}$

ウ．$\frac{C_0x}{r}+\frac{C_1r}{x}$　エ．$\frac{C_0r}{x}+\frac{C_1x}{2}$

オ．$\frac{C_0x}{2}+\frac{C_1r}{x}$

〈d〜fに関する解答群〉

ア．$\sqrt[3]{\frac{C_0r^2}{C_1}}$　イ．$\sqrt{\frac{C_1}{C_0r}}$

ウ．$\sqrt{\frac{2C_0r}{C_1}}$　エ．$\sqrt{\frac{2C_1r}{C_0}}$

オ．$\sqrt[2]{\frac{C_0}{C_1r}}$　カ．$\sqrt{\frac{2C_0}{C_1r}}$

キ．$\sqrt{\frac{2C_1}{C_0r}}$　ク．$\frac{3}{2}\sqrt[3]{C_0^2C_1r}$

ケ．$\sqrt{2C_0C_1}$　コ．$\sqrt{2C_0C_1r}$

平成元年　秋期　第2種　問18

解　説

在庫管理にかかる総費用は発注費用と保管費用の和として計算されます．

総費用＝発注費用＋保管費用

まず，発注費用について考えてみましょう．

発注費用は発注回数に1回当たりの発注経費をかけて求めることができます．さらに，発注回数は年間の総需要量を発注量で割って求めることができます．簡単にするために発注経費は発注量に無関係に一定であるとすると発注費用は次の式で計算することができます．

発注費用＝(総需要量／発注量)×(発注経費)

次に年間の保管費用について考えてみましょう．ここで，在庫が同じ割合で減少していくとすると図3においてAの部分の面積とBの部分の面積は等しくなります．在庫の保管費用は保管時期には無関係ですから，Aの部分の在庫をBの部分で保管しても保管費用に変化のないことがわかります．これより**図4**と**図5**の保管費用が等しくなることも明らかでしょう．図5において在庫量は年間を通して同じですからこの数に在庫1個当たりの年間保管経費をかけた値が保管費用ということになります．したがって保管費用は次の式で計算されることになります．

保管費用＝(1/2)×(発注量)×(在庫1個当たりの年間保管経費)

ここで，発注量以外はすべて定数であるならば，総費用は発注量の関数ということになります．このとき総費用を最小にする発注量は総費用を発注量で微分した式を0にする発注量として求めることができます．ただし，微分の知識のない人でもこの場合は次のようにして総費用を最小にする発注量を計算することができます．

発注量が2倍になると発注回数は1/2倍になり発注費用も1/2倍になります．つまり，発注費用は発注量に反比例しているわけです．これに対して，発注量が2倍になると平均在庫量は2倍になり，保管費用も2倍になります．つまり，保管費用は発注量に比例していることがわかります．総費用，発注費用，保管費用を1つの図に書き込むと発注費用と保管費用を等しくする発注量が総費用を最小にする発注量であることがわかります．この費用は最適発注量，もしくは，経済的発注量と呼ばれています（**図6**）．

解　答

a－オ，　b－ア，　c－エ，

d－ウ，　e－カ，　f－コ

EXERCISE 2

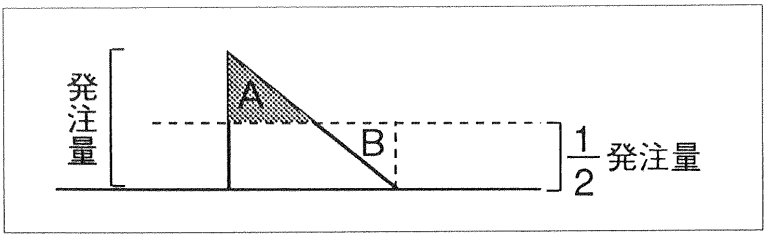
経済的ロットサイズの算定に関する記述中の［　　］に入れるべき適当な字句を解答群の中から選べ．

資材購入の経済的ロットサイズは，

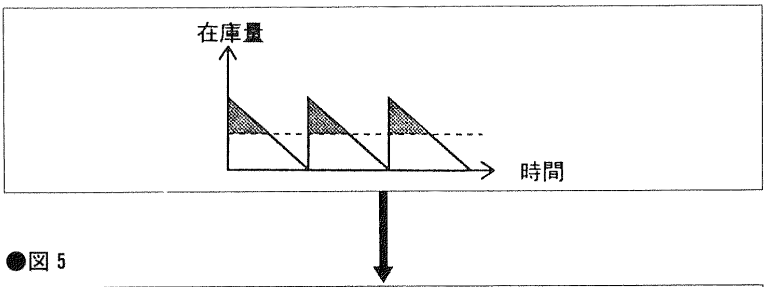




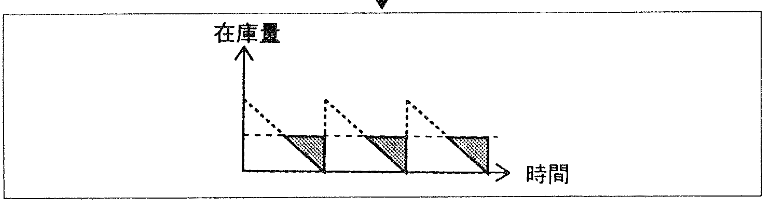
●図 3



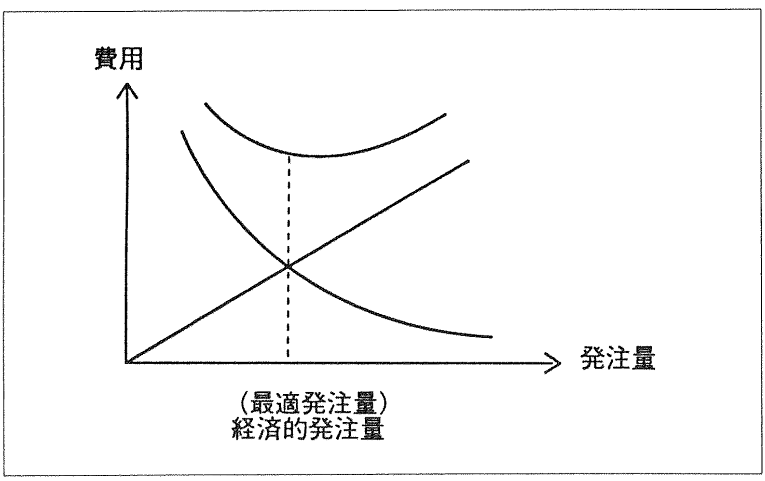
●図 4



●図 5



●図 6



以下のような過程で算出する．

ある品物を１年間に A 円だけ必要とする場合に，それを N 回に分けて購入すると，１回あたりの購入額（ロットサイズ）は A/N 円となる．そして，その品物の在庫がなくなるとまた購入するということを繰り返す場合には，在庫品に対する維持費率を１年間 C で表すと，

在庫維持費＝［ a ］(円)

となる．一方購入のために直接必要な費用が毎回 E 円ずつかかるものとすれば，

［ b ］$=E\cdot N$ (円)

となるので，両者を合計すれば，年間の総費用Tが得られる．すなわち，

$T=$［ c ］(円)

これより，T を最小にする N を求めるには，［ d ］$=0$ として，$N=$［ e ］が経済的ロットサイズとなる．

〈a に対する解答群〉

ア．$\frac{A}{N}\cdot\frac{1}{C}$　　イ．$\frac{A}{N}\cdot C$

ウ．$\frac{1}{2}\cdot\frac{A}{N}\cdot C$　　エ．$\frac{1}{4}\cdot\frac{A}{N}\cdot C$

〈b に対する解答群〉

ア．在庫費用　イ．年間の購入経費

ウ．発注経費　エ．在庫調整費

〈c に対する解答群〉

ア．$\frac{A}{N\cdot C}+E\cdot N$　イ．$\frac{A\cdot C}{N}+E\cdot N$

ウ．$\frac{AC}{2N}+E\cdot N$　エ．$\frac{1}{4}\cdot\frac{A}{N}\cdot C$

〈d に対する解答群〉

ア．$\frac{dT}{dN}$　イ．$\lim_{n\to\infty}T$　ウ．$\sum_{i=1}^{N}T_i$

〈e に対する解答群〉

ア．$\sqrt{\frac{A\cdot C}{E}}$　　イ．$\sqrt{\frac{A\cdot E}{C}}$

ウ．$\sqrt{\frac{A}{2E\cdot C}}$　　エ．$\sqrt{\frac{A\cdot C}{2E}}$

オ．$\frac{1}{2}\sqrt{\frac{A}{E\cdot C}}$　　カ．$\frac{1}{2}\sqrt{\frac{A\cdot C}{E}}$

昭和54年　問17

※ポイント

発注量がA/Nとなり，円で表されています．この問題では発注量ではなく，発注回数を変数としている点に注意しましょう．

解　答

a－ウ，b－イ，c－ウ，d－ア

e－エ

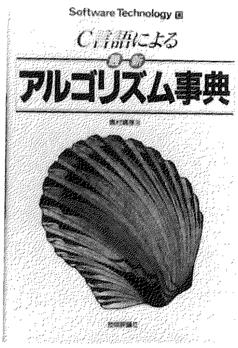
Book Review

■中村正三郎

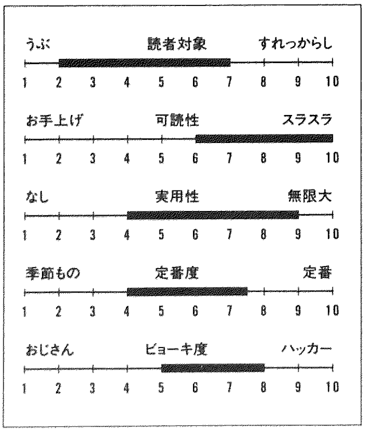
用語事典としても楽しめる説明のおもしろさ

■C言語による最新アルゴリズム事典

奥村晴彦 著
技術評論社　2400円



■評価チャート



本書は，同じ著者による前作「コンピュータアルゴリズム事典」の全面改訂版である．

著者のまえがきによれば，前作は著者の理想の第1近似で，本書はそれよりもさらにいい近似になり得たのではないかという．このへんになると，著者の感覚領域だから，なんともいい難い．

本としては，前作とは全然趣が違う本に仕上がっている．まず Pascal でアルゴリズムを記述していたのを，Cの記述にしてある．これはまあ，どうということはない．大きかったのは，分野別に分類がなされていた前作とは違い，今度はあいうえお順の構成になっていることだ．これが成功しているかどうかは意見が分かれるところだろうが，評者個人は，あいうえお順もなかなか引きやすいと感じた．

全体の構成はといえば，まず，あいうえお順の項目が並び，その後，アルファベット順の項目が並んでおり，最後に索引となっている．

索引には，その言葉が登場するページ番号がいくつか書かれているのは当然として，その言葉を項目見出しとして詳しく説明してあるページ番号は，太字になっているなど，工夫がある．

説明の本文中にも別項目を参照したほうがよいものについては，印がついており，さらにたどっていけるようになっている．こういう工夫を見ると，ハイパーテキスト風に次から次へと引いていけるディスクを添付してほしかったと思う．もちろん別売でもけっこうである．次の機会には考慮していただきたい．

各項目は，見出し，英語表記，説明というパターンになっている．説明は簡潔で要を得ている．また，多くの場合，Cによるプログラムが添えてある．プログラムは，実際に動かしてアルゴリズムに対する理解を深めてもらう点に主眼があると感じた．そのまま引き写して使えるものではない．だが，これは本書の価値を損うものではない．現実の問題に，あるアルゴリズムを適用するには，色々と変更を加える必要があるのは当たり前のことだし，あまり特定の問題に密着した場合，かえって応用がきかないからだ．なお，ソースコードは，技術評論社からディスクとして，あるいはネットワークから入手可能とのことである．

参考文献も示してあり，元の論文を調べたり，関連した話題について調べたりする場合の便宜が図ってある．

先に評者は，あいうえお順も引きやすいと書いたが，これは本書が用語事典としても使えるからだ．実際評者は，用語事典として使用しており，この場合，あいうえお順が引きやすい．つまり，こういうことをしたいのだがどういうアルゴリズムがあるかを調べるための事典としては使っていないのだ．おそらく著者の意に反した邪悪な使い方であろうが，便利だから仕方ない．

パズルのアルゴリズムを収集しているのも本書の特徴で，たとえば，小町算とはどういうものだったかなどは，す

ワンラインNEWS
ジャストシステムはデータベースソフトを開発，秋にも発表する．出荷は来春の予定．データベースには競合製品が多いが，「一太郎」との連携利用を売り物にする．

ぐわかる．そんなところをうれしがっているのである．

さらに白状すれば，古今東西のあらゆるアルゴリズムを集大成することをめざしている著者には申し訳ないが，評者にとっての本書は暇潰しにもってこいの本になっている．暇なときに，ぱらぱらめくって，たまたま開いたページを読み始めると簡明な説明がおもしろく，さらに関連項目も読んで，といった具合になかなか止められないのである．

DOS 5.0登場で注目されるMS-DOS関連書籍

古田島義和（神田・書泉グランデ）

米国マイクロソフト社は，91年6月に「DOS 5.0」の出荷開始を発表しました．現在，全世界で3000万台以上あるといわれる「DOS 4.0」にかわり，ファイル・ミラーリング・バックアップなどのユーティリティを搭載，メモリ管理強化やWindows性能の向上などを実現して，大幅にバージョンアップしました．

2，3年前までは，OS/2がDOSに代わって次世代OSとして普及していくだろうとの見方が大勢を占めていましたが，Windowsの出現でOS/2への期待が薄れ，今年「DOS 5.0」の登場でMS-DOSへの期待が高まってきました．また，日本語版「DOS 5.0」も10月には出荷の見通しであると発表されています．これによって，日本の各パソコンメーカーの4.0に対する評価の低さをいっきに挽回できるでしょう．

一方，IBMのDOS/VもVer. 4からVer. 5.0へ，同じころに移行する予定で，パソコン分野で苦労してきたIBMにとって，明るい材料です．出版物もソフトバンクの「DOS/V PC事典」が発売時から大好評で，今後のVer. 5が製品化された時が楽しみです．

そのほかのDOS関連で注目の図書を少し紹介します．Addison Wesley (以下AW) の「Undocumented DOS」(A. Schulmanほか共著) が90年11月以来，大好評でたびたび品切れになるくらいです．これは，DOS機能の隠された部分を集大成した本で，著者の中には，実際に開発に参加した人も含まれていて，内容的にとてもグレードの高いものです．ほかにRay Duncan著のMS-DOS関連書籍が人気になっています．Microsoft Pressの「The MS-DOS Encyclopedia」(アスキーから訳本あり)，「Advanced MS-DOS programming 2/E」「MS-DOS Functions programmer's Quick Reference」，AWの「Extending DOS」が代表的なものです．

そしてDOS 5.0については，Dvorak's著のものがユーザーに支持されています．McGraw Hill (以下MH) の「Dvorak's Guide to DOS and PC performance」，Bantamの「Dvorak's Guide to PC Connectivity」があります．そのほかにBantamの「Devoney's DOS 5 Guide」(C. DeVoney著)，同社「DOS Power Tools(5.0対応)」，入門書としては，MHの「Simply DOS」(K. Jamsa著) が最適でしょう．

新刊では，Prentice Hall (以下PH) から「C++:A Guide for C Programmer's」(ソフトバンクから訳本あり) や，「Lisp」などで人気の高いS. Hekmatpour氏が「Templa Graphica (Mac Version)」を出版．Macユーザーにとても好評のようです．PHの「Fractal programming and Ray Tracing with C++」(R. T. Stevens著) も好評で，最近，Fractal関連が幅広く売れています．W. H. Freemanの「Fractals Chaos power Laws」や，同社の「Fractals: (Video)」などがあります．

最後に，今後注目される新刊を紹介します．M&Tの「Delivering E-Mail」，「Troubleshooting LAN Manager 2.0」，AWの「System 7 Revealed」，「The Macintosh Companion」，「An Introduction to Computer Graphics Concepts」と続々出版されます．

■和書&洋書ベスト10

(書泉グランデ調べ：1991年5月1日～5月31日)

1	サン・マイクロシステムズ	アスキー	1	Compilers (paper)	Addison Wesley
2	C言語による最新アルゴリズム事典	技術評論社	2	Programming perl	O'Reilly
3	プログラミング言語C　第2版	共立出版	3	Programming Windows 2/E	Microsoft Press
4	UNIXユーティリティライブラリmake	啓学出版	4	System performance Tuning	O'Reilly
5	たのしいUNIX	アスキー	5	ResEdit Reference Ver.2.1	Prentice Hall
6	SUNシステム管理	アスキー	6	Object-Oriented Analysis 2/E	Prentice Hall
7	パソコン見栄講座	翔泳社	7	Mac Scheme:Manual and Software	M.I.T.
8	コンサルタントの秘密	共立出版	8	The Metafont Book	Addison Wesley
9	人工現実感の世界	工業調査会	9	Undocumented DOS	Addison Wesley
10	入門C言語　改訂新版	アスキー	10	Simple Book	Prentice Hall

ワンラインNEWS
ASTリサーチ・ジャパンはOADGに加盟する意向を明らかにした．同社は昨年，日本電気「PC-9800シリーズ」と米IBM「PC/AT」の両互換機を発売している．

◆◆ 佐々木哲のニューヨーク ◆◆

N.Y. システム通信

Windows 3.1情報

この夏よりWindows 3.1のベータ版の配布がはじまるようです．もっともTrueTypeが装備される以外は大きな変更はなく，おもにより使いやすく，安定したバージョンアップという意味あいのようです．Windowsの世界は，しばらくは大きな変更もなく，サードパーティの活躍が期待されるところです．開発ツールもVisual-Basic，Borland C＋＋など手軽なものがそろってきました．

数万色の同時発色が可能なRAM/DACチップ

さて，先月からの続きです．高解像度化，Windowsに特化した描画機能をもった超高速Super VGAチップなどVGAの発展はとどまるところを知りません．今月は少し多色化の話をしましょう．

オリジナルのVGAは320×200の解像度で256色を65,000色のパレットから選んで表示するものでしたが，最近のSuperVGAでは1MBのメモリをもち，1024×768の解像度で同じく256色の表示ができるものも増えましたが，いくら解像度が増えても，同時表示できる色数は8ビットのままです．高価で特殊なディスプレイボードでなく，VGAでもっと色の数を増やして自然な画像が表示できないものでしょうか．

VGAカードには最後の段階でデジタルデータをアナログに変換して出力するためのD-Aコンバータ（DAC）がチップとしてのっています．ここではパレットの処理も行われます．

これを差し替えるための，高機能なRAM/DACチップが2社から発表されています．ひとつがEdsun LaboratoriesのCEG/DAC，もうひとつがSierra SemiconductorのHiColor Palette chipです．どちらも既存のVGAに使われているRAM/DACとピンコンパチブルで，極端な話，VGAボードのRAM/DACをこれに差し替えるだけで，数万色の同時表示が可能になります．

両者にはコンセプトの違いがあり，Edsunは特殊なストリームをVRAMに書き込むことによりダイナミックなパレットの制御を行い，74万色の同時表示を可能にしています．さらにグラフィック画面においてリアルタイムのスムージング機能（Continuous Edge Graphics）をもっています．ただしプログラミングが複雑で比較的動作が遅いのが難点のようです．

一方，Sierraは，TARGAとも互換性をもつ15ビットの32,768色の同時表示を速度の低下なしで行えます．また比較的チップの値段も安いようです．

とはいえ，どちらも数10ドルのチップです．すでにどちらもWindowsドライバ，Autocadドライバなどが用意されており，ソフトウェアの負担もありません．実際に300ドルほどで，このような高解像度，多色のSuper VGAが手に入るわけです．

いまのところ，このチップを使ったSuper VGAボードも出荷がはじまったばかりですが，OrchidのProDesigner II sのようにRAM/DACをわざわざソケットにしてつけ，オプションで提供しているものもあります．

偉大なI/O，プリンタポート

IBM PCの偉大さはその拡張性とコネクティビティ（connectivity）にあります．その拡張性を支えてきたのはいわゆるISAバス，PC/AT拡張バスですが，PS/2のいささか閉じたアーキテクチャ，Laptopマシンなどの拡張性に欠けるパソコンの増加もあり，また一般ユーザーにとっても，ケースのふたをあけて，拡張スロットにインストールするのは面倒なことです．もっと手軽なI/Oはないでしょうか？

いまもっともホットなI/O，それはプリンタポートです．本来はプリンタをつなぐためのパラレルポートですが，オリジナルのIBM PC以来，直接プログラムしやすい構造になっています．またIBM ATモデル339のころから出力だけでなく，入力も許す双方向のものになりました．比較的高速なパラレルのデータの出入口，これを放っておく手はありません．日本でもJ-3100用にEasyhardという偉大な（しかも無料の！）アイデアがありました．今月はプリンタポートにつながるものを集めて紹介してみましょう．

・SCSIホストアダプタ

そろそろ日本でもおなじみのTrantorのminiSCSIです.昨年秋紹介されたこの製品はHD, CDROM, MOなどのSCSIデバイスをプリンタポートに接続するものです.昔から低価格なSCSIホストアダプタを各種販売してきたTrantorだけに,ソフトウェアドライバもMSCDEXをはじめ豊富に付属しています.プリンタも同時につなげます.残念ながら入力は,シリアル変換してステータスラインから行っているため少し遅め.

・テープドライブ

TEAC製のカセットテープサイズのテープを使ったものがあります. ValitekのPST-60, PST-160*1はそれぞれ60MB, 160Mバイトのデータを記録できるポータブルのテープドライブです.重さ9.5lb(5kg弱)のドライブユニットをパラレルポートにつなぎ,専用ソフトで安心してバックアップを取ることができます.転送速度も386機で1分間4.5MBの速さです.

・ハードディスク

Notebook, Laptop用にポータブルのものが数社から出ています. Disk Technologies CorpのRoadRunner RXDシリーズ*2は2.5inchのHDを使い, 20MB, 60MBの容量.当然パラレルポートにつながります.専用のドライバによりDOSのドライブとして使えます.

・フロッピーディスク

5インチ, 3.5インチのフロッピーディスクが飛び交う現代のオフィスでは,結局どの機械にも両サイズのドライブが欲しくなります.簡単にどの機械のプリンタポートにでもつながるドライブが最適.たとえばTimatic SystemsのBackpack*3があります.製品には3.5インチ 720k/1.44Mのもの,最近IBMが採用した2.88MBもサポートしたもの, 5.25インチのものがあります.

・ベルヌーリディスク

ユニークなリムーバルディスクであるIomegaのベルヌーリディスクも,パラレルポートにつながる携帯用のものがあります.もともと1,000Gにも耐えられる衝撃に強い構造ですからますます最適です. Transportable 44*4は44MBのベルヌーリドライブがひとつ収まった外部ユニットです. SCSIデバイスなのですが,パラレルポートにつながるアダプタも提供しています.

・スキャナ

Epsonをはじめ数社がパラレルポートにつながるスキャナを発売しています.スキャナは大量のビットマップデータを読み込むため,高速なインターフェースが必要です. RS-232Cでは遅すぎます.といってGP-IBなどのインターフェースボードを用意するのも面倒です.簡単にプリンタポートにつなぐだけで快適に文書や絵を読み込めます.

・ファイル転送

Travelling SoftwareのLapLink 3が代表格ですね. 2台のパソコンのプリンタポートを特殊なケーブルでつなぎ,高速にファイルを転送します. PS/2とともに3.5 inch FDが広く紹介されたときにはIBM自身も同様な製品を出しました.

・LAN

これはXircomがPocket LAN adapter*5で開拓した分野ですね.いまでは10Base-T, Token ring, Arcnetといったさまざまな形態のもの, Netwareにとどまらず各種NOSに対応した製品が各社から出ています.値段も400ドルを切るものも出ています.

・キーボード

そう,プリンタポートにつながるキーボードです. Notebookパソコンなどではキーボードが打ちにくいのが難です.自分の家にいるときくらい大きなキーボードを使いたいもの.ところがキーボードコネクタが付いていないNotebookパソコンではどうするか. Genovation, Inc.のParallel Keyboard*6の出番です. 101キーのフルサイズ,テンキーだけのモデルがあります.付属のドライバで,アプリケーションからは普通のキーボードとして使うことができます.

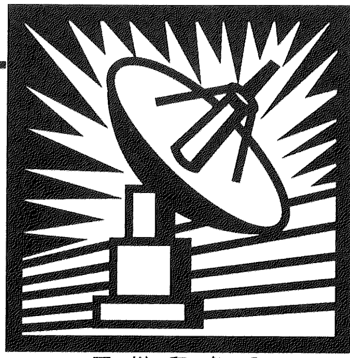
*1 PST-60 $1,695/PST-160 $1,995
Valitek, Mountain Farms Mall,Hadley, MA 01035, phone (415) 586-7408
*2 Disk Technologies Corp., 147 W. Lyman Ave., Ste. 200 Winter Park, FL 32789, phone (800) 553-0337, (407) 645-0001, fax (407) 628-4404
*3 Backpack 1200 5.25inch 2D/2HD unit. $349.00
Micro Solutions Computer Products 132 W. Lincoln Hwy. DeKalb, IL 60115, phone (815) 756-3411, fax (815) 756-2928
*4 Bernoulli Transportable 44 $1,259 Iomega Corp. 1821 West 4000 South Roy, UT 84067, phone (801) 778-1000, fax (801) 778-3450
*5 Pocket LAN Adapter $595.00 for Ethernet, Xircom, Inc. 22231 Mulholland Hwy., Ste. 114, Woodland Hills, CA 91364, phone (818) 884-8755, fax (818) 884-1719
*6 Parallel Keyboard 101 $200.00, Genovation, Inc., 17741 Mitchell, N, Irvine, CA 92714, phone (714) 833-3355, fax (714) 833-0322

DB25ピンコネクタを使った当たり前のパラレルポート.実にいろいろなものがつながります.まだ何かありそうですね.あなたは何をつなげたいですか? なお,この柔軟性の高いポート,当然ながらプリンタもつながります.

今月のなんだこりゃ?

もっと何か驚くものが知りたいのですか? 来月も乞うご期待!

Illustration : NOBUKI FUKUHARA



西川和久の
希望的観測

DOS 5.0登場!!
「省メモリ」「タスクスイッチング」など機能満載

1 6月11日，噂のDOS 5.0ついに出荷開始 なんと！マスターディスクからは起動しない

DOS互換であるDR-DOSが，先にDOS 5.0で実現されるであろう機能をほとんど搭載してDR-DOS 5.0を出荷してしまい，後追いのかたちでDOS 5.0が米国で6月11日に発表され，即日出荷開始された．PC-DOS，COMPAQ-DOSなど，OEM先を含むすべてのDOSがいっせいに5.0になったのだ．

以前からの噂通り，「メモリ環境の改善」「タスクスイッチング」など，DOS 4.0から比べると大幅な機能アップがなされている．日本国内でDOS 4.0は，メモリ環境上あまり使われなかったので，各メーカーともひさびさの大幅なDOSバージョンアップが行われることであろう．

このDOS 5.0，これまでとは少し変わっていて，マスターディスク中にはブートローダーが含まれていない．1枚目のディスクを差し込んでリセットしても，DOSは立ち上がらないのだ．バージョンは何でもいいのだが，従来のDOSがあるディスク上に必要ファイルをロードして，アップデートするかたちになっている．

インストールの方法は，WindowsでおなじみのSETUP.EXEを起動するかたちで，メッセージに従い3枚(2DDの場合)のディスクを順番に入れコピー終了後，リセットするとDOS 5.0が立ち上がるしかけだ．

このとき，いままで使っていたDOSのファイルは，“¥OLD_DOS.1”というディレクトリに保存されていて，必要であればUNINSTALLコマンドを使い，いつでも古いVersionが復活できるよう工夫されている．当然，古いバージョンを削除するコマンド，DELOLDOS.EXEも含まれる．

2 最大限630KBのフリーエリアを確保 フルスクリーンのタスクスイッチャ機能も

メモリ環境は，劇的に改善された．CPUの能力に合わせてDOSをチューニングできるようになっているのだ．

第1の改善点は，DOSのカーネルサイズが4.0のときより小さくなったこと．バージョンアップすると，サイズは大きくなってしまうのが世の常であるが，DOS 5.0に関しては例外なのだ．

第2の改善点は，80286以上のCPUであれば，DOSのカーネルやバッファをHMAと呼ばれるエリアへロードすることにより，主メモリへの圧迫をより小さくできること．さらに，80386 (SX) 以上のCPUでは，デバイスドライバや常駐ソフトをUMBへロードする機能，すなわち，Memory-Pro386やQEMM-386を使って得られる効果をDOSレベルでサポートしてしまったのだ．

これらの機能を最大限に利用すると，なんとフリーエリアが630KB以上確保できる．日本語FEPなどでメモリを圧迫されているわが国のパソコン環境にはうってつけのDOSだといえよう．

ワンラインNEWS
東芝は米国などで販売する自社パソコン「Tシリーズ」で日本語ソフトが利用できるようにする「日本語エミュレータ」を開発，米国で販売を開始した．

また,「Windowsはタスクスイッチャだ!!」と強調しているメーカーがあるが, DOS 5.0の出現で考え方を改めなければならない. DosShellと呼ばれるメニュープログラムから,フルスクリーン切り替えではあるが,タスクスイッチングをDOSレベルでサポートしたのだ. わざわざ, アプリケーションスイッチのためにWindowsを買う必要がなくなってしまった.

3 やっとついたオンラインヘルプ BASIC,ツール類などおまけも満載

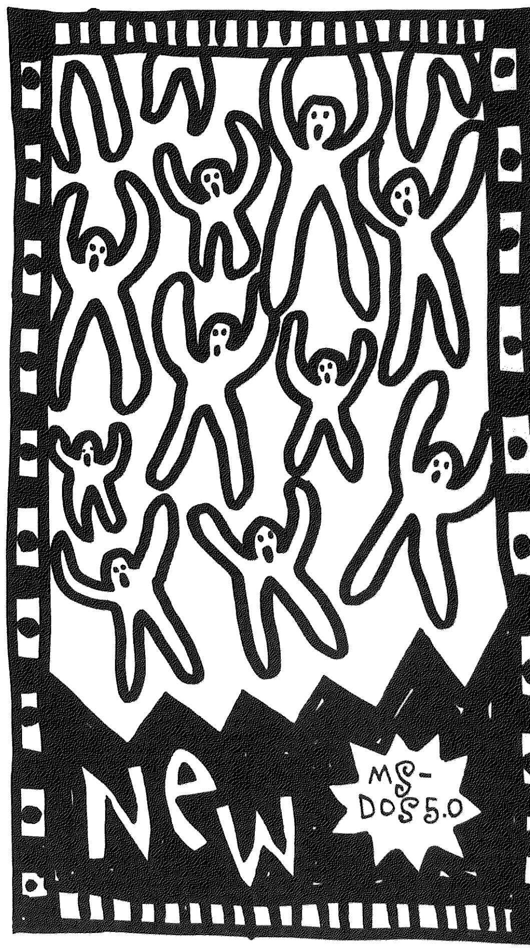
各コマンドの機能強化にともない, いろいろなオプションが追加されている. いつも使うコマンドなら覚えているが, たまにしか使わないコマンドは忘れてしまって, マニュアルを読むことはよくある話だ. DOS 5.0に付属するすべてのコマンドで, “/?”オプションを付けると, 簡単な使い方が画面上に表示される.「やっと付いたか!!」という感もあるが, これは非常に便利である.

DOS 5.0は,ツールの大安売りである. QuickBASIC, Screen Editor, UnDelete, Mirror, LoadFix, DosKey, UnFormat, EMM386, SetVer, Net5など, これまで市販ソフトに含まれていたツール, アプリケーションが山のように入っている. なかでも, インタープリタ型のQiuckBASICまでが標準装備だというのには驚いてしまった. もちろん, ゲームやデモなど, 簡単なサンプルプログラムも含まれている.

4 DOS 5.0を簡単にDOS/V化!? 日本語FEPもちゃんと動く

番外編として, 簡易DOS 5.0/V (?) を作る方法があるので紹介しておこう. 最初に述べたように, DOS 5.0では従来のDOSに対してSETUPコマンドでアップデート行うため, PC-DOSの拡張DOSでもあるDOS/Vでは, 英語モードに切り替えてSETUPを実行すると, 即, DOS 5.0/Vのできあがりである. 画面, キーボード回りの問題で, KEYBOARD.SYS, KEYB.COM, COUNTRY.SYS, ANSI.SYSは, DOS/Vのファイルをもう一度 “¥DOS” へコピーしなおす必要があるが, いとも簡単にできあがってしまう.

「日本語環境でタスクスイッチすると画面が乱れる」「ファイル/ディレクトリ名に全角は使えない」など, 多少問題があるものの, 日本語FEPを含むすべてのアプリケーションは問題なく作動する. メモリ環境などを考えると非常においしい話である. たまに, DOSのバージョンを調べているため起動しないものがあるが, DOS 5.0ではSETVERコマンドがあり, そのようなプログラムに対してファイル単位で疑似的にDOSのバージョンが変更できるため心配無用である.



Computer Laws

コンピュータの法律

プログラムの改造・その2

◆

熊谷信太郎(弁護士)

■著作権では認められている改造

今回は，前回に引き続いてプログラムの改造に関する諸問題をとりあげたいと思います．

市販のプログラムを買ってきたら，パッケージに，改造禁止の特約が記載されている場合がありますが，この改造禁止の特約は有効なのでしょうか．

前回に説明しましたように，「特定の電子計算機においては利用しえないプログラムの著作物を当該電子計算機において利用しうるようにするため，又はプログラムの著作物を電子計算機においてより効果的に利用しうるようにするために必要な改変」については，プログラム著作者の同一性保持権は及ばないとされています*1．

このような法律の規定からすると，「より効果的に利用」するため，または，「コンピュータで利用できるようにするため」のプログラムの改造は大幅に認めるのが，著作権法の原則だということができます．

■特約は有効か

設例の場合，このような原則を修正する特約ということになりますが，この特約が有効かどうかは，結局，著作権法が定める，同一性保持権の例外として，プログラムの改造を認める規定が強行規定なのかどうかによって決まります．

もし強行規定ということになると，その法律の規定と異なる内容の特約を当事者がしても，特約は無効になり，法律の規定どおりの効果が発生しますが，任意規定ならば，法律の規定と異なる内容の特約も有効になります．

強行規定か任意規定かという問題は，著作権法の分野に限ったことではなく，たとえば，高利でお金を貸す場合の利息の上限は利息制限法で規制されていますが，お金を借りる際に，「利息制限法の上限を越える利息を支払う」旨の特約をすれば，その特約は有効なのかという場合にも同様に問題になります．

もともと利息制限法は，借主の弱い立場につけ込んで，不当に高利の貸付がなされることを放置すると，高利貸しが不当な暴利を得る一方で，高金利の負担に耐えられない借主の生活が破綻するなどの社会問題が発生しかねないので，そのような事態を防ぐという立法趣旨で制定されたのですから，利息制限法の規定を潜脱するような特約は無効にすべきであって，その意味で，利息制限法は強行規定であると考えられています．

■著作権法は任意規定か強行規定か

では，「より効果的に利用」するため，または，「コンピュータで利用できるようにするため」のプログラムの改造を大幅に認めている著作権法の規定は強行規定なのでしょうか．

これについては，考え方が分かれていますが，任意規定と考えるのが大勢を占めているといっていいでしょう．つまり，プログラム著作権者の保護のために，プログラムの改造を自由にはさせないという特約を無効にするまでの利益が認められないと考えられているのです．そして，そのような判断の背景には，プログラムの改造を制限する特約を結ぶのが一般化しているという，現在の業界の実態があるのです．

市販のプログラムのパッケージに，改造を制限する条項が印刷されていて，パッケージを開封すると，その条

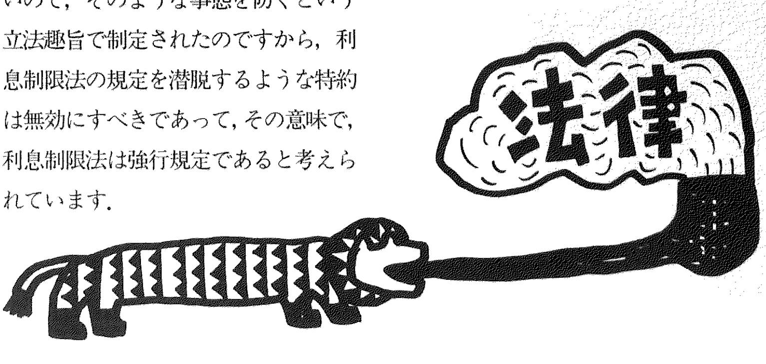


Illustration: Nobuki Fukuhara

項を承諾したものとみなすという文言が記載されていることがありますが，これはまさに，プログラムの改造を認めている著作権法の規定を排除するための，当事者間の特約の申し出にほかならないのです．そして，特約が成立するためには，申し出と承諾がいるのですが，パッケージを破るのが，特約の申し出に対する承諾になり，その時点でプログラムの改造を認めている著作権法の規定を排除する特約が成立するのです．

次回は，そのような特約があるのに，プログラムを改造してしまった場合はどうなるのかを検討したいと思います．

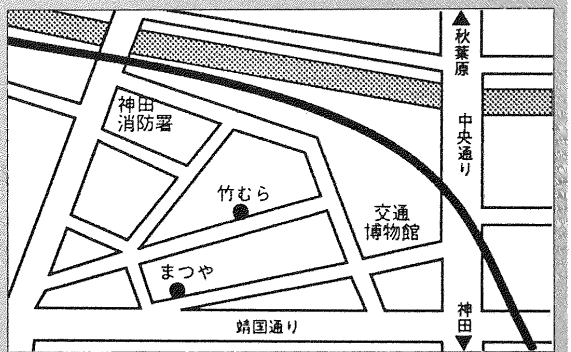
＊1　著作権法第20条第２項第３号

アキバ短信

まつや

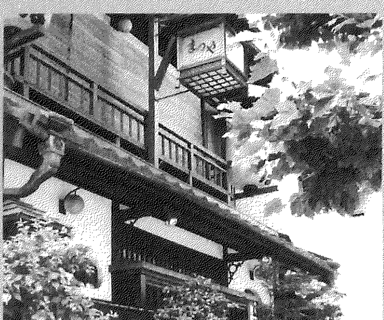
アキバ帰りに粋を味わう

営業時間　11：00～20：00　日曜定休
TEL 03-3251-1556



秋葉原電気街にはいくつかのアクセス方法がある．もっともわかりやすいのは，JR山手線・京浜東北線と総武線のクロスする秋葉原駅で下車して「電気街口」へ出ることだが，意外に地下鉄の便もよい．日比谷線「秋葉原」，銀座線「末広町」，丸の内線「淡路町」，都営新宿線「岩本町」「小川町」の各駅が，ぐるりと秋葉原電気街を取り囲んでいる．それぞれの駅から電気街まではせいぜい２～３分だが，その数分で思わぬところに発見があるかもしれない．

淡路町の交差点，神田消防署，交通博物館を結んだ三角地帯は古い町並みが残っていて，隣接する秋葉原とはまた違った雰囲気がただよう．このあたりは有名な飲食店が多いことでも知られている．店員さんが独特のかけ声（？）で迎えてくれる「藪蕎麦」，鳥すきの「ぼたん」，うなぎの「中川」，洋食の「松栄亭」など，いずれおとらぬ老舗ぞろいだ．


◀まつや


▼竹むら

「まつや」もこういった老舗のひとつ，淡路町駅から交通博物館へ向かう通りの左側にある蕎麦やである．純和風の店構え，２階から下がった「手打そば」と書かれた提灯，加えてお昼時には店の外まで順番を待つ人があふれているから，まず見過ごすことはない．

おすすめは何といっても「ごまそば」(580円)．もりそばのそばつゆが胡麻風味になったものを想像してもらえばいいのだが，胡麻の香りが手打ちの細いそばによく合って，しかも意外にあっさりしていてうまい．もっとしっかり食べたいという人には，かきあげ天丼(800円)などどんぶりものもある．買い物の帰りならば，そばミソやにしんをつまみに一献傾けるのも悪くない．「藪蕎麦」に比べると知名度の点でやや劣るが，秋葉原で痩せたサイフをそれほど痛めることなく味わえるのがいい．

甘党ならすぐ近くにあるおしるこの「竹むら」へ寄って，桜湯であげまんじゅうをいただくのもいいだろう．ただし，このあたりの店はまつやに限らず日曜はほとんどが休みなので要注意．まぁ，ひとりで秋葉原を散策した帰りにでも，ちょっと立ち寄るのが粋というものだろう．

ワンラインNEWS
YHPはアップルの「Macintosh」に対応したプリンタ「HPディスクライター」を約３割値下げした．アップル社が３月に低価格プリンタを市場に投入したことに対応する．

●プログラマの仕事場 | 5

Programmer's Working Space

株式会社アドミラルシステム

ハードウェア，BIOSへの依存部分を切りわけることによってどの機種へもプログラムの移植は可能

アドミラルシステムは，'80年代半ばに，IBM PC/ATを使ったファクトリーオートメーションでスタートした会社だ．その後，使用機種はJ-3100，AX，PC-9800シリーズなど多様なものになったが，その過程で機種間のポータビリティの高いプログラムを開発する技術を蓄積してきた．そうした技術の成果が，「Newton-98」などのユニークなパッケージソフトとして登場している．埼玉県川口市に同社を訪ねて，同社企画室室長の黒岩潤司氏にお話をうかがった．

工場の街 川口で自動化システムからスタート

高機能，低価格のディスクユーティリティソフト「Newton-98」が話題のアドミラルシステムだが，設立当初は工場関係が中心だったという．「ここ川口は鋳物の街で，むかし吉永小百合の『キューポラのある街』という映画の舞台にもなったことがあるんです．ですから電子部品とか，コンピュータとはあまり関係ありませんが，工場がけっこうありましたので，工場の自動化のお手伝いをするような関係でスタートしたわけです．あるメーカーの半導体製造装置をやっていたという資料がいまでも残っています．いわゆるファクトリーオートメーションからスタートしたんですね」．

「84年ですと，当時はまだマイコンベースです．いまでいうところのICEを使っていました．そのときのホストがATで，ワープロ以外では98を使ったことのないというめずらしい会社でしたよ．その後，ある大手のNCメーカー（編注　NC：Numerical Control:コンピュータを使って工作機械を制御すること）の基本開発も手伝ったりして，私たちで作った製品がいろいろなフェアで，そのお客様の名前で出たりもしていました」と黒岩氏は当時をふりかえる．

“ないものは自分で作る”姿勢がJ-3100ユーザーのニーズと合致

PC/ATで開発を行ってきた同社は，東芝が日本でJ-3100を発売すると，そのサイズのコンパクトなこと，東芝から借りたデモ機のテスト結果がまずまずであったことから，社内のATをJ-3100に切り換えた．結果としてこれがパソコンビジネスへの進出のきっかけとなったのだという．

「J-3100が発表になったのが11月で，実際に納品されたのが12月末でした．とりあえずテストだけでもと思って動かすとデモ機と違ってどのソフトも動かない．東芝の本社に話が違うと文句をいうと，デモ機はIBMのPC-DOSで動かしていたけれど，製品に付属のものは日本語MS-DOSだから，OSが違うので動かないというのです．しかも，PC-DOSにあたる英語MS-DOSは，品切れで4月まで手に入らな

い．すったもんだしたあげくなんとか年明けからPC-DOSを手に入れることができました.」

「でも，スタートがこんな具合でしたからトラブルばかり．当時のJ-3100の初期モデルはハードディスクが10MB，フロッピーディスクが2DDのみ．IBMに3.5はインチがないので，いったん98に落とさない限りコンバートできないんですが，98の2DDをもってくると読めない．それじゃ外付けの5インチを付けようと注文を出すと4月まではないというんです．テストのときに3.5インチに落としてあったソフトと，買ってきた一太郎くらいしか使えませんでした.」

「次に困ったのが，『印字ができない』プログラムのリストをとろうとしても，日本語のメッセージがあると出ない．東芝では『漢字プリンタを買ってくれ』という．それが，50万くらいするんですよ．『動かない』『ハードディスクに何も入れられない』『入ったとしても印字ができない』仕方がないので，社内用に作ったのがこの3つ，『RSCOPY』『MAGIC RUN』『プリンタデバイスドライバ』です.」

「RSCOPY」は，シリアルポート経由で，ファイルを高速で転送するユーティリティ．まずはこれを使って，J-3100のハードディスクをいっぱいにする．次に，「MAGIC RUN」で日本語DOS上でPC-DOSをエミュレーションしてアプリケーションを動かす．印刷には「プリンタデバイスドライバ」を使えば，IBMの標準であるESC/Pをサポートするプリンタに打ち出せる．3つのユーティリティを使ってどうやらJ―3100で開発ができるようになった．黒岩氏は，ファクトリーオートメーションの世界では，ないものは自分で作るのはあたりまえだし，最初はまったく売りに出すつもりなどなかったと語る．ところが，同社の様子をおそるおそるうかがいに来た東芝の営業マンから「ぜひほかのユーザーにも提供してほしい」と強くすすめられて，パッケージとして販売することになったのだそうだ．

ソフトウェアのレイヤー化で開発後の機種間移植はわずか2週間

IBM PC互換機としてAXが登場したときも，パッケージ化こそしなかったものの，社内では各社AXマシン用のJ-3100エミュレータを開発して，Lotus 1-2-3など既存のJ-3100アプリケーションを利用していたという．

開発環境のメインはPC互換機だが，マーケットとしてのPC-9800シリーズには早くから注目し，電子手帳とパソコンのデータ転送ユーティリティである「パソコン電子手帳くん」などのソフトも開発している．「9800シリーズ用はブローダーバンドさんに販売をお願いしましたので，ウチの製品だということがあまり知られませんでした」(黒岩氏)．という経緯もあり，また，機能的にIBM PC用ソフトに負けないよいものをと，自社ブランドで出したのが「Newton-98」である．

同社が続々とこうした製品を送り出していける背景には，かつてMS-DOSの移植に何度か携わり，MS-DOSについては隅々までよく理解しているということ，そしてファクトリーオートメーションの分野で培った高い技術力がある．自社製品の機種間の移植などの依頼を受けると，たいてい2週間くらいで基本部分はなんとかな

About Company

丸山治昭社長．1984年の設立当初は，工場の自動化を支援するプログラムの開発などが中心だった．現在はBIOS，OSをはじめとするソフトウェア開発，マイクロチップ，電子回路，コンピュータシステム応用機器の設計と開発，コンサルティング業務など幅広く手がける．「Newton-98」をはじめとするパソコン用のパッケージソフトは売上の約50パーセントを占める．社員12名，そのうち6名が技術者である．



Location

埼玉県川口市飯塚．川口駅西口を出て，徒歩約10分ほど．今年5月に完成したばかりのビルにオフィスを構える．周辺には「キューポラの町」の象徴である鋳物工場も見られる．



企画室室長　黒岩潤司氏

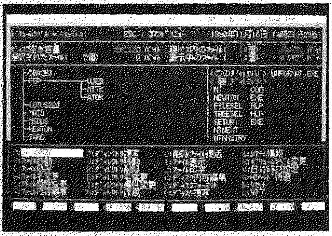
Products

「パソコン電子手帳くん」(J-3100用14,800円 PC-9800シリーズ用12,800円) シャープの電子手帳と，パソコンを接続するためのソフトウェアと専用ケーブルのセット．電子手帳のフォーマットと一致したパソコンの入力画面で，データを入力，電子手帳に送信したり，電子手帳のデータをパソコン側で受信，MS-DOSファイルとして保存することも可能．NEC PC-9800シリ

ーズ，東芝J-3100シリーズ対応版がある．

「MAGIC RUN PLUS」(19,800円)
「RSCOPY」(30,000円)
「MAGIC PRINT」(30,000円)
それぞれ，東芝J-3100シリーズの日本語MS-DOSでPC用アプリケーションを使うためのPC-DOSエミュレータ，異機種間，異なるメディア間でデータ交換をするためのファイル転送ユーティリティ，東芝J-3100シリーズでIBM製プリンタを利用するためのプリントユーティリティ．いずれも最初は自社内で業務上の必要にせまられて作ったソフトウェアであった．

「Newton-98」(9,700円)
ファイルやディレクトリの選択による，コピー，移動，削除，削除ファイル，ディレクトリの復活やファイル再配置によるディスクアクセスの高速化，ヒストリー付きのビジュアルシェル機能など20あまりの機能をもつディスク管理ユーティリティソフト．5インチ2HDと3.5インチ2DDがひとつのパッケージに含まれている．NEC PC-9800シリーズ用．


Newton-98

Modula-2

Pascalを作ったことで知られるスイスのNiklaus Wirth教授が，パーソナルワークステーションLilithのために考案したプログラミング言語．その経緯からもわかるように，ハードウェアを直接操作するような低レベルの記述からアプリケーションプログラムに必要な抽象化された記述までが可能である．
並列処理を実現するための機能を言語仕様レベルで備えている，モジュール化が徹底されている，強力な型チェックを行うなどの特徴がある．

ってしまうという．

「私たちはずっと米国のIBMのマシン（と互換機）で開発をしてきましたし，MS-DOSの移植もしましたので，実にいろいろなソースを見てきています．それで気づいたのですが，ほとんどのプログラムではDOSとハードウェアの切り分けをしていません．日本以外の国ではパソコンといえば，IBM PCとマッキントッシュしかないわけでしょう．これはハードに依存する部分，これはマシンに依存する部分，これはDOSに依存する部分だなどと区別する必要などないので，すべていっしょくたに書いてしまうのです．この傾向は日本の98だけで育ったエンジニアにもいえます．つまり，「移植層」のような，通信でいうところのレイヤー的な発想がないのです．」

「われわれは，移植の仕事もしていましたから，こんなものの移植を1からやらされる羽目になってはたいへんだから早めに手をつけておこうと，機種ごとにレイヤーの切り分けをしたんです．マシンごとにスペックに合わせて下側を取り替えられるようなドライバを作りました．DOS依存部分，BIOS依存部分，マシンのハードウェア依存部分というのを分けて作っておくわけです．ウチの製品は『Newton-98』にしても『電子手帳くん』にしても，ハードウェア依存部分とBIOS依存部分はほぼ同じです．新しく開発するのはDOS依存部分だけでいい．そのときでも，ハードウェアを直接さわるようなコードは絶対にプログラマには書かせません．だから，どれかひとつのマシンで動けば，あとはリンクしなおすだけでほとんど動いてしまいます」(黒岩氏)．

プログラミング言語はModula-2，ライブラリは自社で作成

アドミラルシステムではC言語はあまり使わないという．「メインの言語はModula-2です．ファクトリーをやっていた関係で，マルチタスクが最初から必須条件だったのです．MS-DOS上でWindowsも何もなしにマルチタスクができるというと，Modulaくらいしかないですから」(黒岩氏)．同社はModula-2処理系の移植作業にも携わったことがあり，Modula-2に関しては日本でも有数のエキスパートである．現在数種類のModula-2コンパイラを使っているが，ライブラリについては，その内容を把握しておく必要があることから自社作成のものを使っているとのことだ．

日本ではあまり普及していないModula-2のこと，プログラマ探しには，かなり苦労するのではと黒岩氏にたずねたところ「プログラムが書けるかどうかは，経験のある人の場合以外関係ないですね．何よりちゃんと日本語を話せること，あいさつができること，手紙が書けることがたいせつです．それさえできれば立派にプログラマになれますよ（笑）」との返事があった．

IBM PCと互換機での開発が中心のアドミラルシステム．共通のプラットフォームを提供するWindows3.0や，PC互換機でも日本語処理が可能となるDOS/Vなどが続々と登場する新しい環境は願ってもないことだという．今後同社がどのような製品を提供してくれるのかたいへん楽しみだといえよう．

●イベント

■7月24日(水)～7月26日(金)
'91教育総合展
教育関連のハード・ソフトの展示，セミナーも開催される．
会場：科学技術館(東京) 入場料：500円
問い合わせ先：日本教育新聞社 TEL 03-3461-7734

■7月29日(月)～7月31日(水)
第4回土木・建築コンピューター国際会議
土木・建築におけるコンピュータ利用に関する最新技術の国際会議．
展示会も同時に開催される．
会場：池袋サンシャインシティ・コンベンションセンターTOKYO，池袋サンシャインシティ・ミプロ国際展示会場(東京)
参加費：55,000円(一般，同伴家族は27,500円)，10,000円(博士コースまでの学生)
問い合わせ先：日本エクスポマネジメント(株)内運営事務局 TEL 03-3239-1144

■9月4日(水)～9月6日(金)
データベース'91 TOKYO
商用データベース，インハウス・データベース用の各種システムの展示．
国際シンポジウムも開催される．
会場：池袋サンシャインシティ・センターホールA(東京)
入場料：500円
問い合わせ先：日本経済新聞社事業局 TEL 03-3252-8157

■9月5日(木)～9月8日(日)
1991年電子情報通信学会秋季大会
一般講演と通信・エレクトロニクスなどのシンポジウム講演が行われる．
会場：玉川大学(東京・町田市)
参加費：1,000円(会員)，1,700円(非会員)，500円(学生)
問い合わせ先：(社)電子情報通信学会 TEL 03-3433-6691

■9月25日(水)～9月27日(金)
第6回IC—光カード&ASCAT展
ICカード，光カードの最新技術と関連機器の展示．
セミナー，「SCAN-TECH JAPAN '91」も同時開催される．
会場：東京流通センター・第2展示場(東京・平和島) 入場料：1,000円
問い合わせ先：日本工業新聞社事業部 TEL 03-3292-3561

●セミナー

■8月5日(月)～8月7日(水)
オブジェクト指向設計とC++セミナー
オブジェクト指向技術による
C++プログラミングを体験しながら学習する．
会場：麹町会館(東京)／受講料：145,000円(3名以上割引あ
問い合わせ先：ソフトウェアテクノロジーインターナショ
TEL 03-5689-3536

information・page

Microsoft University

■7月23日(火)～7月26日(金)
Microsoft LAN Manager Programming Environment
LANマネージャアプリケーション開発のための入門コース．
会場：特別会場T 受講料：150,000円(4日間)

■7月23日(火)～7月24日(水)，8月20日(火)～8月21日(水)
C言語入門
Quick Cを使ったC言語の入門コース．
会場：特別会場T 受講料：60,000円(2日間)

■7月29日(月)～8月2日(金)，8月26日(月)～8月30日(金)
MS OS/2 Presentation Manager Applications Programming
OS/2 PMアプリケーション開発のための上級コース．
会場：特別会場T 受講料：150,000円(5日間)

■7月29日(月)～8月2日(金)，8月26日(月)～8月30日(金)
Microsoft Windows Applications Programming
Windowsアプリケーション開発のための上級コース．
会場：三田会場 受講料：180,000円(5日間)

■8月5日(月)～8月9日(金)，8月19日(月)～8月23日(金)
9月2日(月)～9月6日(金)
Microsoft Windows Programming Environment
Windowsアプリケーション開発のための入門コース．
会場：三田会場，特別会場T 受講料：150,000円(5日間)

■8月5日(月)～8月9日(金)
MS OS/2 Applications Programming
OS/2 APIを使った実習中心のコース．
会場：三田会場 受講料：150,000円(5日間)

■8月7日(水)～8月9日(金)
C言語応用
MS-C 6.0を使ってC言語の高度な技術を学ぶ実習中心のコース．
会場：特別会場T 受講料：90,000円(3日間)

■8月19日(月)～8月23日(金)
MS OS/2 Presentation Manager Programming Environment
OS/2 PMアプリケーション開発のための基本コース．
会場：特別会場T 受講料：150,000円(5日間)

■8月26日(月)～8月30日(金)
Microsoft LAN Manager Applications Programming
LANマネージャアプリケーション開発のための上級コース．
会場：特別会場T 受講料：180,000円(5日間)

Microsoft Universityは，OS/2，Windowsアプリケーション開発者のために
定期的に開催されているセミナーです．
問い合わせ先：富士ソフトウエア(株)MU教育センター TEL 03-3452-7722

NEWS & NEW PRODUCTS

ボーランドジャパン，Winアプリ開発が可能な「BORLAND C++ 2.0」を発売

（株）ボーランドジャパン（TEL 03-3486-1400）は，SDK（Software Development Kit）を使わずにWindows3.0用のアプリケーション開発が行える「BORLAND C++ 2.0」を発売した．C++をサポートした「BORLAND C++ 2.0」は，ソフトウェア開発における生産性と信頼性を向上させるオブジェクト指向プログラミングが可能で，TURBOシリーズの統合環境とコンパイルスピードの速さを受け継ぎ，Windows 3.0アプリケーション開発に使用することのできる「TURBO DEBUGGER」「TURBO ASSEMBLER」，ダイアログ，アイコン，ビットマップなどの編集を可能にした「リソースエディタ」などのツール類を標準装備している．また，TURBO DRIVEによるプロテクトモードでのコンパイル，2度目のコンパイルの際に一度コンパイルしたヘッダを使用することによりコンパイルスピードを格段に向上させるリコンパイルヘッダ機能などを備えている．

■おもな仕様

対応機種：PC-9801シリーズ（ハイレゾモード対応，LTを除く）／必要メモリ：640KB以上（TURBO DRIVEは1MBの拡張メモリ）／OS：MS-DOS 3.1以上，MS-Windows 3.0以上／周辺機器：ハードディスク（必須），マウス・EMSに対応／価格：98,000円

マイクロソフト，DOS/V初の言語製品PS/55用MS-C 6.0を発売

マイクロソフト（株）（インフォメーションセンター：TEL 03-3363-1201）は，DOS/Vに対応したIBMパーソナルシステム／55（PS/55）シリーズ用「Microsoft C Professional Development System Version 6.0」（以下MS-C 6.0）を開発し，8月下旬から日本アイ・ビー・エム（株）を通じて発売する．これは同社初のDOS/V対応言語ソフトで，これによりすでに日本アイ・ビー・エムから発売されている「日本語Microsoft Windowsソフトウェア開発支援キット」とあわせて利用することにより，PS/55シリーズでのWindowsアプリケーションの開発が可能となった．

MS-C 6.0は，プログラマの生産性を大きく向上させる新しい開発環境「プログラマーズ・ワークベンチ（PWB）」を装備し，デバッグ機能とユーザーインターフェースを改善し，大容量・高速実行を実現した「コード・ビュー」などを備え，OS/2用に対応したDLLとマルチスレッドをサポートしている．また，DOS/Vに対応したグラフィックスライブラリとBIOSライブラリも提供される．

■おもな仕様

対応機種：IBM PS/55シリーズ／動作環境：IBM DOSバージョンJ4.0，IBM DOSバージョンJ4.0/V，必要メモリ640KB以上またはIBM OS/2バージョン1.23以上，空きメモリ1MB以上，最小8MB，最大23MBの空き領域のあるハードディスク（PWB使用時は約9MB，DOSのみの選択時約15MB，OS/2のみ選択時約20MB）／価格：98,000円

ライフボート，4GBのメモリ空間を使用できるDOSエクステンダーを発売

（株）ライフボート（TEL 03-3293-4711）は，MS-DOSの640KBのメモリ制限を取りはらい，CPUに80386を搭載したマシンで最大4GBのメモリ空間を使用することのできるDOSエクステンダー「DOS/4G」を発売した．メモリ領域に関する業界標準規格DPMIに準拠しているため，Windows 3.0などとの互換性をもつ．オプションの仮想記憶マネージャVMMを使用することにより，実装メモリが不確定なパッケージアプリケーションの開発が可能となる．価格は，シングルユーザーライセンス版が880,000円（VMM 580,000円），最大20台のマシンで仕様できるコーポレートライセンス版が2,640,000円（VMM 1,740,000円）．

ワンラインNEWS
富士通はインテルの32ビットCPU「80386SX（16MHz）」を搭載したブック型パソコン「FM R50NBX」を発売した．

NEWS & NEW PRODUCTS

吉正電子，同期式をサポートしたRS-232Cライブラリを発売

吉正電子(株)(PC事業部：TEL 03-5371-3041)は，C言語でRS-232Cの各種制御を行うためのライブラリ「RS-232C Library」を発売した．これは一般的な調歩同期式だけでなく，同期式を完全にサポートしており，BSCなどの同期式手順を使うアプリケーションにも対応することができる．半／全二重制御，ハード／ソフトフロー制御，拡張RS-232Cボード（PC-9861K）をサポートし，すべてのライブラリには，日本語コメント付きのCソースコードが添付されており，アプリケーションソフトへの組み込みの際のロイヤリティも不要．また，プログラム例としてだけでなく，実際に使用することのできる通信ソフト「Y-TERM」が付属している．価格は，23,000円で，MS-C Ver.5.10以上，QuickC Ver.2.00以上に対応している．

エプソン，ハイレゾモード対応の低価格32ビットマシンを発売

セイコーエプソン(株)，エプソン販売(株)（エプソンPCインフォメーションセンターTEL 03-3377-3531)は，ハイレゾモードを標準装備し，30万円台の低価格を実現した32ビットパソコン「エプソンPC-386GS」とオプションのハイレゾボード（39,800円）の装着によりハイレゾモード対応が可能な「同PC-386GE」を発売した．両機種とも，CPUボード上に増設RAMモジュールの直接接載が可能で，EGC互換の拡張グラフィック描画機能，サウンド機能などを標準装備している．

■エプソンPC-386GS

プロセッサ：80386（20MHz）／メモリ：1.6MB，ハイレゾモード時1.5MB（最大14.6MB，ハイレゾモード時14.5MB）／外部記憶装置：3.5インチFDD×2（GS-2），3.5インチFDD×2・100MB HDD×1（GS-2E），3.5インチFDD×2・5インチFDD×1（GS-3），5インチFDD×2（GS-5），5インチFDD×2・100MB HDD×1（GS-5E）／価格：398,000円（GS-2/GS-5），423,000円（GS-3），598,000円（GS-2E/GS-5E）

■エプソンPC386-GE

プロセッサ：80386SX（16MHz）／メモリ：1.6MB，ハイレゾモード時1.5MB（最大14.6MB，ハイレゾモード時14.5MB）／外部記憶装置：3.5インチFDD×2（GE-2），3.5インチFDD×2・40MB HDD×1（GE-2B），3.5インチFDD×2・5インチFDD×1（GE-3），5インチFDD×2（GE-5），5インチFDD×2・40MB HDD×1（GE-5B）／価格：298,000円（GE-2/GE-5），323,000円（GE-3），428,000円（GE-2B/GE-5B）

東芝，386SX/20MHz，プラズマディスプレイ搭載のDynaBookを発売

（株）東芝（パソコン営業推進部：TEL 03-3457-2930）は，CPUに80386SX（20MHz）を搭載し，プラズマディスプレイを採用した「DynaBook 386/20 J-3100SXシリーズ」を発売した．従来機に比べて約1.25倍の処理速度を実現し，液晶に比べて約10万倍の応答速度で，ちらつきがなく視野角の広いプラズマディスプレイを，ブック型パソコンとしては世界ではじめて採用した．RS-232Cインターフェースを2チャンネル標準装備し，マウスとモデムなどを同時に使用することができる．世界時計・カレンダー・電卓などをメニューから呼び出すことのできる「新DynaBookメニュー」を採用している．また，着脱可能で，データのセキュリティ，持ち運び，保存が容易なハードディスクパック（20，40，60MB）も用意されている．

■ DynaBook386/20 J-3100SX 001P/021P/041P/061Pモデル

プロセッサ：80386SX（20MHz））／メモリ：2MB（最大10MBまで拡張可能）／外部記憶装置：3.5インチFDD×1（001P），20MB HDD×1（021P），40MB HDD×1（041P），60MB HDD×1（061P）（HDDはハードディスクパックと交換可能）／

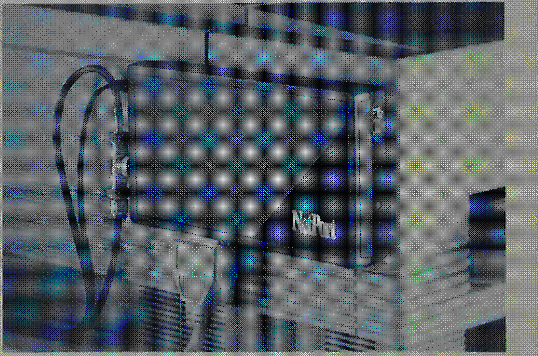
NEWS & NEW PRODUCTS

ディスプレイ：プラズマディスプレイ／価格：298,000円(001P)，398,000円(021P)，438,000円(041P)，498,000円(061P)

ソニー，書き換え可能な3.5インチ光磁気ディスクシステムを商品化

ソニー(株)(コンピューター周辺機器営業本部データストレージ国内営業部：03-3448-3444)は，国際標準規格(ISO)に準拠した3.5インチ光磁気ディスクドライブを商品化し，サンプル出荷を開始した．3.5インチMOディスクドライブ「SMO-301」シリーズは，1分間のディスク回転数を3000回転と，業界最高速のデータの読み出し／書き込み速度を実現している．サイズは，底面積が3.5インチフロッピーディスクドライブと同じサイズで，コンピュータ機器への組み込みが容易となっている．価格は，3.5インチMOディスクドライブ「SMO-D301」，同ディスクドライブコントローラー「SMO-C301」が300,000円(サンプル価格)，コントローラー内蔵3.5インチMOディスクドライブ「SMO-P301」が300,000円(サンプル価格)．また，128MBの大容量記憶が可能な3.5インチMOディスク「EDM-128」(価格：7,900円)も発売された．

インテルジャパン，NetWare386J対応のプリントサーバを発表



インテルジャパン(株)(TEL 0298-47-8511)は，ノベル社のネットワークOS，NetWare386Jに対応したネットワークプリントサーバ「NetPort」2機種を発表した．VHSビデオカセットテープ大のコンパクトサイズで，プリントサーバ用のパソコンを介さずに，直接プリンタをイーサネットケーブルに接続することができる．また，シリアル，パラレルポートを1つずつもっているため，同時に2台のプリンタをサポートすることができる．価格は，134,000円(予価)．

コンテック，小規模パソコンLAN用ネットワークOSを発売



(株)コンテック(TEL 03-3769-1061)は，5台から10台程度の小規模LANを主対象としたパソコンLAN OS「LANジュニア」(PC-9800，98noteシリーズ用)を発売した．「LANジュニア」は，すべてのパソコンを対等に使うことのできるピアツーピア型(対等方式)で，サーバ専用機が不要．メモリ常駐量がワークステーションで6KBと非常に少なく，各種の設定も簡単なメニュー方式で行うことができる．OS単体価格は，1ネットワークあたり198,000円で，パソコンを10台接続した場合，1台あたり20,000円弱と安価なネットワーク構築が可能．テスト導入のために必要なハード・

ワンラインNEWS
三菱電機はマルチスキャン方式の新型ディスプレイを発売．日本語Windows 3.0の要求する表示仕様に合わせるため640×400から640×480ドットに自動的に切り替える．

NEWS & NEW PRODUCTS

ソフトをセットした試用セット「エントリパック」（デスクトップ２台用／174,000円，デスクトップ＋98note／196,000円）も用意されている．

富士通，TOWNSのマルチメディア機能を大幅強化する製品群を発売

富士通（株）（FM TOWNSインフォメーションサービス：TEL 03-5690-0032）は，FM TOWNS用CD-ROM XAの開発環境やマルチメディアプレゼンテーションツールなど，マルチメディア機能を大幅に強化するための製品群を発売した．CD-ROM XA開発環境として，音声の圧縮記録を行う「ADPCM録音カード」（248,000円），「ADPCM開発支援ライブラリV1.1」（98,000円），圧縮音声と画像の同時再生を行う「ADPCM再生カード」（34,800円），「CD-ROM XA共通ライブラリV1.1」（24,000円）．マルチプレゼンテーションツールとして，写真，ビデオ映像，音声などのマルチメディアを利用したプレゼンテーションの作成から実行までを簡単に行うことのできるオーサリングツール「伝」（70,000円）．また，ソニーの電子ブック（EB仕様）の検索も行えるワープロソフト「FM-OASYS V1.0（LBP版）」（61,000円），VTRやレーザーディスクなどの映像を鮮明に再生（1677万色表示），パソコン画面との重ね合わせ表示を実現する「ビデオプロセッサカード」（198,000円，10月末出荷）と「ビデオライブラリ V1.1」（68,000円，10月末出荷）を発売した．

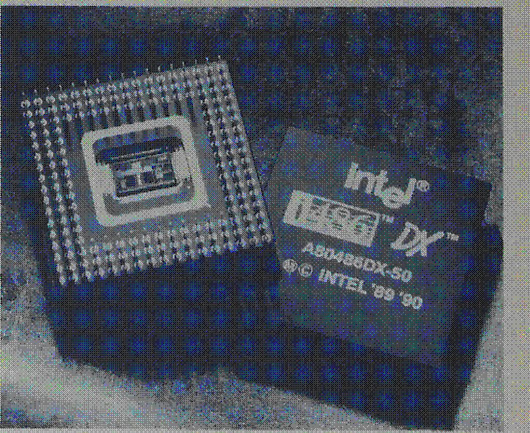
メルコ，98DA，RA用内部増設RAMボードを低価格で発売

（株）メルコ（メルコインフォメーションセンター：TEL 052-251-8365）は，PC-9801DA，RA21/51用の内部増設RAMボード「MDA-4000」を発売した．これは，4MBで49,800円の低価格を実現し，標準サブボード増設方式を採用し，メモリ部は増設RAMサブボード「XMC-2000L」で構成されているためすべて取り外しが可能で，H98などのほかの機種へメモリの継承ができる．仮想86EMS，UMB，RAMディスク，プリンタスプーラなどの各種設定が可能なメモリ統合管理ソフト「MELWARE」も付属している．

また，PC-9801DX用内部増設RAMボード「EDXシリーズ」も発売した．メモリ未実装タイプの「EDX-0」（9,800円），「同2000」（2MB，34,800円），「同4000」（4MB，59,800円），「同8000」（8MB，110,000円）がある．

インテルジャパン，50MHzの486マイクロプロセッサを発表

インテルジャパン（株）（TEL 0298-47-8511）は，従来の33MHz版にくらべて50％の高速化を実現した50MHz動作の「Intel486™ DXマイクロプロセッサ」と新しい二次キャッシュメモリコンポーネント（82495DX/82490DX）を集積したテスト済みの50MHzモジュール「Intel486™ DX CPUキャッシュ・モジュール」を発表した．MS-DOS，Windows 3.0，UNIXベースのソフトウェアとの互換性を備え，グラフィックスやネットワークといった高速で大量のデータ処理を行うシステムでの採用を目標としている．また，三層メタルCHMOS Vプロセス技術により，従来の486CPUにくらべて約半分のチップサイズを実現している．



ワンラインNEWS
米AMDは米インテル社互換MPUの新機種「Am386SX」の販売を開始した．同社は先に米インテル社のMPU「i386」の互換MPUを販売しているが，日本では伸び悩んでいる．

Programmer's Program

人気のアセンブラソースジェネレータ 日本語マシンに対応 SOURCER(ソーサー)

「SOURCER」はIBM社のPCおよびその互換機上で動作する，絶大な人気を誇る逆アセンブラで，本誌5月号の特集中でも紹介している．

これまでPC-9801やJ-3100などの国産機種では，IBM PCエミュレータを使うか，英語DOS上でなければ動作しなかったが，今回ライフボート(株)から日本語MS-DOS対応版としてSOURCER VersionJ3.18が発売された．対応機種は，日本電気PC-9800シリーズ，富士通FMRシリーズ，東芝J-3100シリーズ，日本IBM PS/55シリーズ，各社AXとなっている．

SOURCERの機能

SOURCERのおもな機能を以下にあげる．

- 8086/87/8088，80186/188，80286/287，80386/387，80486，V20/30の命令セット実行プログラムの解析，逆アセンブルが可能．
- MASM Ver.4.0/5.0/5.1，TASM Ver.1.0/2.0，OPTASM Ver.1.0/1.5の各アセンブラ処理系に対応した逆アセンブラソースリストの作成が可能．
- COMファイル，デバイスドライバ，EXEファイル，オーバーレイファイル，SYSやBINなどのバイナリファイル，メモリ上にすでにアロケートされているプログラム（ROM領域も含む）の各実行形式プログラムに対して，ユーザーが指定する任意のアドレス範囲の逆アセンブルが可能．
- 複雑に入り混じるマルチセグメント構成に対して，最大250個までのセグメントの解析が可能．
- 標準で5パス，最大9パスまでの綿密な解析逆アセンブル処理を行う．
- 解析結果の逆アセンブルソースリストには，使用されているファンクションコールの機能と各レジスタの意味，アクセスしているデータ領域に関する細かい情報，I/Oポートの操作状況，複雑なコード処理内容などについての，詳細なドキュメントに類するコメントが付加される．
- 解析オプションによって，解析方法をユーザー側で細かく設定できる．さらに，定義ファイルを使用することによって，逆アセンブルソースリストのカスタマイズも行える．

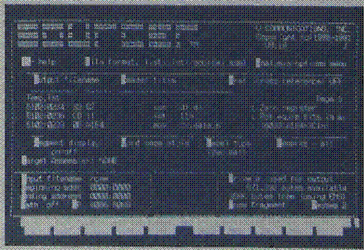
ほかにも，次のような機能がある．自動的にCALLやJMP命令の関連を解析しラベルを付加する．定義ファイルを使用し，好みのラベル名やコードのカスタマイズが可能．メモリ常駐型プログラムの割り込みエントリアドレスを認識することが可能．アイテムがどこで定義されているかを調べるためのクロスリファレンス情報の出力．プレフィックスニーモニックを正確に解析．自動的にデバイスドライバを判断し，エントリポイントを認識．アセンブラのバージョンの違いを考慮した逆アセンブルが可能．

モノクロ表示だが機能は変わらず

パッケージには，5インチまたは3.5インチの2HDフロッピーディスク1枚と約120ページのマニュアル1冊，ユーザー登録ハガキなどが入っている．ハードディスクへのインストールは，マスターディスクを挿入し，MS-DOSのコマンドラインから，

```
A> README
```

と入力する．あとは，インフォメーションプログラムのメインメニューからインストールを選択し，指示にしたがって操作すれば，作業は完了する．このインフォメーションプログラムを用いて，基本操作の説明やデモンストレーション，マニュアルに記載されてい



ない機能の説明などを見ることができる.

SOURCERの起動は,インストールしたディレクトリに移動するか,PATHの通ったディレクトリにSOURCERをおき,MS-DOSのコマンドラインから

```
A> SR target.ext -parameter
```

と入力する(target.extは解析するプログラムのファイル名).起動パラメータには-j(全角データを区別する),-d(デフォルトの定義ファイルを使わない),-e(EMSメモリを使わない)などを指定することができる.

ファイル名を省略して起動する場合はIコマンドで解析するファイルを指定する.起動画面は,エスケープシーケンスキャラクタのみを使ったモノクロ画面である.IBM PC用SOURCERのカラーを使った美しい画面に比べると,かなりの違和感があるが,SOURCERとしての実力はなんら変わらない.起動画面で各種コマンドを指定し,[G]キーで解析の開始である.

なお,SOURCERはEMSメモリがあれば,プログラムのロードやシンボルテーブルにEMSメモリを割り当てるので,より巨大なプログラムを解析することができるようになる.

処理系を指定して出力できる

メニューから指定できるコマンドは図1の通りである.さらに[A]コマンドによって細かなオプションの指定をすることができる.

オプションには,コード領域とデータ領域の区別方法,割り込み,ポート操作,ジャンプ命令,データの参照範囲の制限など,複雑なプログラムの解析をするための機能が用意されている.

解析されたアセンブラソースは,任意の処理系に依存した形式で出力でき,もちろんアセンブラを用いて再アセンブルすることが可能である.解析可能なCPUは,8088から80486までのすべてのCPUに対応し,数値演算コプロセッサ用のコードも解析できる.

生成されたソースには,使用されているファンクションリクエストの機能と各レジスタの意味,I/Oポートの機能,複雑なコード処理内容などについての詳細なドキュメントに類するコメントが付加される.

同時に生成される定義ファイルには,解析時に使用した各種コマンドやオプションがテキストファイルとして記録され,同じコードをSOURCERで逆アセンブルするときに自動的にロードされる.

＊

SOURCERを使用すれば,あらゆるMS-DOSの実行形式ファイルから,処理内容に対応した豊富なコメントを伴った読みやすいアセンブルソースを作成することが可能である.市販ソフトウェアなど,ソースコードが公開されていないプログラムのデバッグをしたいときなど,手軽にプログラムの内部情報を得ることが可能になる.

解析されたアセンブルソースは,8088から80486までの全命令をサポートし,しかもMASM,TASMなどターゲットのアセンブラを指定することによって,そのまま再アセンブルが可能である.

SOURCERは,このように強力かつ正確なアセンブラソースジェネレータであるが,全角データが挿入されたコードの解析にはやはり多少無理があるようである.

●SOURCER

販売:(株)ライフボート(TEL 03-3293-4711)/価格:38,000円/動作機種:日本電気PC-9800シリーズ,富士通FMRシリーズ,東芝J-3100シリーズ,松下電器産業PanacomMシリーズ,日本IBM PS/55シリーズ,各社AX/動作環境:MS-DOS Ver.2.X以上/必要メモリ:512KB以上/供給メディア:5インチまたは3.5インチフロッピーディスク/日本語マニュアル

図1　SOURCERのメインメニューコマンド

[?]	ヘルプ画面をオープンし,コマンドの説明を表示
[G]	解析の開始
[ESC],[Q]	SOURCERの終了
[F]	出力ファイル形式の選択
[H]	リストファイル出力用のヘッダ(タイトル)の指定
[L]	ラベル形式の選択
[O]	出力ファイル名の指定
[R]	自動的に付加するコメントの形式の選択
[S]	リストファイルのセグメント表示の有無の選択
[W]	命令やラベルの表示の大文字か小文字かの選択
[X]	クロスリファレンスの作成の有無の選択
[A]	解析オプションメニューの表示
[B]	プロセスの開始アドレスの指定(ROMの解析用)
[C]	解析するファイルのコード型の選択
[D]	出力ファイルのドライブ指定
[E]	プロセスの終了アドレスの指定(ROMの解析用)
[I]	解析するプログラムファイル名の指定
[M]	数値演算コプロセッサ用のフィルタの有無の選択
[P]	パス数の選択
[T]	ソースアセンブラの選択
[U]	CPUの選択

オンラインソフト採集帳 8

デバイスドライバを選択するプログラム CINIT.SYS

起動時に単一のドライブから複数のCONFIG.SYSを選択したり，デバイスドライバを一時的に登録したり，登録から外せたらと考えている人は少なくないと思う．CINIT.SYSはこんな要望に応える便利なプログラムである．

CINIT.SYSのインストール

CINIT102.LZHは圧縮されているので，復元には吉崎氏の高機能書庫圧縮管理ユーティリティLHAかLHarcが必要となる．LHAをカレントか環境変数PATHで指定されたディレクトリに準備し，CINIT102.LZHのあるディレクトリに移動する．MS-DOSのコマンドラインから

A> LHA E CINIT102

と入力すれば，CINIT.SYSやCINIT.DOCを含む複数のファイルがカレントに復元される．CINIT.DOCには詳細な利用方法が書かれているので，使用前に十分参照して欲しい．

組み込みはエディタを使って，起動ドライブのCONFIG.SYSの先頭にdevice=CINIT.SYSの1行を加え（もちろんCINIT.SYSをコピーしたパスを指定する必要がある），lastdrive文で複数のデバイスドライバのグループを区切る（リスト1参照）．

本来，lastdriveは，アクセス可能な最後のドライブ名を指定するコマンドである．CINIT.SYSを組み込む場合，このコマンドはlastdrive=endよりあとに記述することで有効となる．

また，ふだん使用しないデバイスドライバは，

device=-A:¥DRV¥RSDRV.SYS

のようにdevice=とファイル名の間に「-」を加えれば，選択画面の中から一時的に登録することができる．

CINIT.SYSの選択画面

CINIT.SYSを組み込んでMS-DOSを立ちあげると，デバイスドライバの選択画面が表示される．この画面は上下2段にわかれていて，上段にファンクションキーとlastdrive文に記述したコメントが表示される．ファンクションキーまたは数字を入力すると，下段に選択されたグループのデバイスドライバが表示される．

ここで［↑］［↓］キーでリバースラインを登録から外してあるデバイスドライバに移動し，スペースキーを押すことで一時的に登録することができる．逆に登録から一時的に外すことも可能だ．さらに［TAB］キーを押せばパラメータの編集をすることもできる．

設定が終わったら［RETURN］キーを押す．選択画面で設定されたデバイスドライバのグループが組み込まれてコマンド入力待ちとなる．

CINIT.SYSのパラメータ

デフォルトでは，つねに起動時に選択画面を表示するが，CINIT.SYSにパラメータを与えることにより，任意のキーが押されていたときのみ選択画面に移行することも可能である．

-KG	([GRPH])
-KK	([カナ])
-KC	([CAPS])
-KS	([SHIFT])
-KT	([CTRL])

これらをオプションパラメータで指定した場合，キーが押されていなければ最初のグループが選択されたことになる．

プログラムからグループを判断

AUTOEXEC.BATからCINIT.SYSで選択されたデバイスドライバ

リスト1 CINIT.SYSを使ったCONFIG.SYSの例

```
FILES   = 50
BUFFERS = 20
device  = B:¥dev¥CINIT.SYS -KC
lastdrive=1 EMM386 (Win3.0) +R23 (Free)  +SMARTDRV (Win3.0)
    device  = -B:¥dev¥DUALDRV.SYS
    device  = B:¥dev¥HIMEM.SYS
    device  = B:¥dev¥EMM386.SYS /M=3072 /F=C000-CFFF
    device  = B:¥dev¥RZ.SYS /U=3 /E=100
    device  = B:¥dev¥SMARTDRV.SYS 2048
    device  = A:¥fep¥VJEB.DRV /SYS=A:¥fep /A: /ml /hs /j /l /ks /kn
    device  = -B:¥dev¥MAXLINK.DRV a: b: c: d:
    shell   = A:¥COMMAND.COM A:¥ /p /e:1024
lastdrive=2 MELEMM (メルコ)  +MELDISK (メルコ)  +MELCACHI (メルコ)
    device  = -B:¥dev¥DUALDRV.SYS
    DEVICE  = B:¥dev¥MELEMM.SYS /NC /GV86 /SW1 /M D0,D8
    DEVICE  = B:¥dev¥MELDISK.SYS E 1664 /S
    DEVICE  = B:¥dev¥MELCACHI.SYS E 2048 /F1 F2 HS1 S
    device  = A:¥fep¥VJEB.DRV /SYS=A:¥fep /A: /ml /hs /j /l /ks /kn
    device  = -B:¥dev¥MAXLINK.DRV a: b: c: d:
    shell   = A:¥COMMAND.COM A:¥ /p /e:1024
lastdrive=3 何も組み込まない
    shell   = A:¥COMMAND.COM A:¥ /p /e:1024
lastdrive=end
```


ワンラインNEWS
ローランドDGはアップルのMacintoshに対応するCG出力用プロッタ「スケッチメイト for Mac」を発売した．8色カラーでの，カッティングシートの加工も可能．

のグループを知ることができる．たとえば，デバイスドライバの選択画面で［f･1］キーが押されたときは$$INIT1，［f･2］キーが押されたときは$$INIT2というように，CINIT.SYSは$$INITx（xは1から9の数字）というデバイスを作成する．

AUTOEXEC.BAT内で

```
IF EXIST $$INIT1 GOTO PART1
IF EXIST $$INIT2 GOTO PART2
```

と記述をすれば，選択されたグループによって分岐先を分けることができる．これを利用すれば自動的に起動する常駐プログラムを変えたり，テンポラリをRAMディスクの有無によって変更することができる．

＊

CINIT.SYSはソースファイルも公開されているので，自分の使いやすいよう改変することも可能だろう．

■CINIT.SYS

分類：デバイスドライバ／ソフト名：CINIT・SYS／登録名：CINIT102.LZH／著作権者：Falcon（ASCII-NET pcs ID:pcs33043）／動作環境：PC-9801シリーズ／入手場所：ASCII-NET pcs（pool msdos）／復元方法：LHA E CINIT102

ファイル構成

Listing of archive : CINIT102.LZH

Name	Original	Packed	Ratio	Date	Time
CINIT.C	15335	4663	30.4%	90-11-22	01:02:00
CINIT.DOC	15320	5096	33.3%	90-11-22	01:02:00
CINIT.FIN	10735	4632	43.1%	90-11-22	01:02:00
CINIT.SYS	13146	3986	30.3%	90-11-22	01:02:00
CONFIG.SMP	1309	347	26.5%	90-11-22	12:33:24
ENTRY.ASM	4666	1737	37.2%	90-11-22	01:02:00
MAKEFILE	389	200	51.4%	90-11-15	16:39:20
README.DOC	1946	1137	58.4%	90-11-22	12:24:08
SAMPLE.BAT	388	99	25.5%	90-11-13	02:41:48
9 files	63234	21897	34.6%	90-11-22	12:43:58

★UPDATE INFORMATION★「DCP」（7月号）のバージョンが3.24に，「FD」（2月号）のバージョンが2.31に

HEALTHY?・UNHEALTHY?

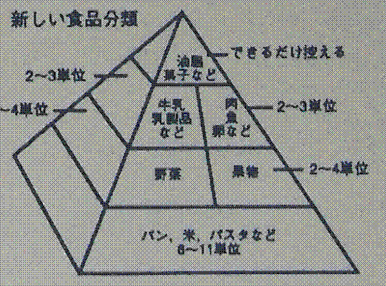
⑧ 米国で提唱された新しい食品分類

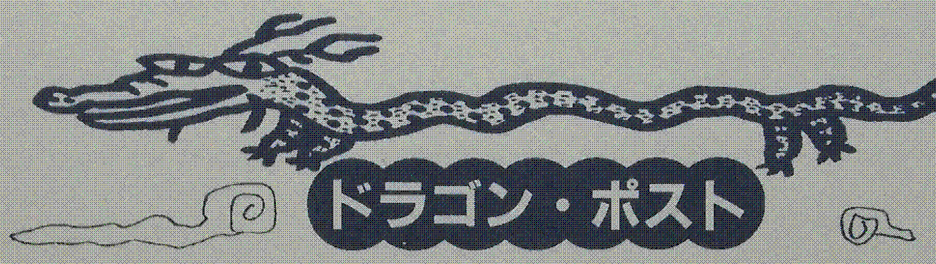
健康食やフィットネスなど，ヘルシーな生活が脚光を浴びるようになってからかなりの年月がたった．こうした話題では，「常識」がよくくつがえされる．かつて，「健康」の象徴であった「生野菜とドレッシングのサラダ」が，野菜はたいして量をとれないし，カロリーの高いサラダ油を使う落第メニューになってしまったように……．

食品摂取の基準であった「4つの食品群」が，いま米国で見直されている．4つとは，牛乳，チーズなどの乳製品，肉・魚・卵類，野菜・果物，そしてパンなどの穀類のことである．1956年に提案されたこの分類法では，乳製品と肉・魚・卵類を各2単位以上，野菜・果物と穀類を各4単位以上とればいいということになっている．

現在，それにかわるものとして提案されているのは，ピラミッド型分類といわれるもの．食生活のベースとなるピラミッドの底辺に穀類を位置づけ，その上に野菜群，果物群，さらに上に乳製品群，肉・魚・卵群，という構成になっている．上になるほど摂取量は少なくなり，ピラミッドの頂点に当たる油脂・菓子類は「できるだけ控える」ということになる．

大きく変わったのは，穀類の扱いである．かつては肥満のもとであるように受け取られていた穀類が，食物繊維を多く含む基本食品としてあげられている．米国での日本食ブームも，米を主食とする食生活への認識が高まったことと無関係ではない．新しい分類を見る限り，穀類を主食として，それに副食を添えるスタイルはさらに注目を浴びることになろう．日本の現状を見ると，逆に以前の米国スタイルの食生活へ傾いているようだ．





ドラゴン・ポスト

Illustration
Nobuki Fukuhara

この欄は，読者からのお便りを中心に構成しています．本誌に対するご意見・ご感想などをお待ちしております．鬼より恐い読者からの声ということで，「ドラゴン・ポスト」です．本誌についているアンケートはがきにご意見の欄を設けてありますから，そこに書いてください．ペンネームでも歓迎です．ただし，はがきには本名と住所をしっかり書いてくださいね．掲載誌をお届けします．

★パソコンとのつきあいが始まったのは……

パソコン博物館のSord Mシリーズが懐かしかった．当時は大学生で，友達がスタートレックゲームやりたさにM-100を買ってからですね，パソコンとのつきあいが始まったのは．テンキーのところにBASICのコマンドキー(LIST，PRINT)があったり，キーボード上にジョイスティックがついていたり，ホビーホビーした楽しいマシンでした．(河口)

――かつて，スタートレックゲームは，本当にスターだったのですね．それのためにマイコンを買ったという人がたくさんいたという話です．

★先にPascalを学べ!?

FORTRAN屋（というほどじゃありませんが）の私には，Cはかなり勝手が違う．あるC-Wizardは「先にPascalを学べ」といっておりましたが，たかが学生にコンパイラ２つも３つも買う金もなし（MS-Cは大学の使ってます），学校では時間が限られる．プレゼントが当たればPascalから始めよう！ダメだったら責任とって「入門C塾」始めてほしいものです．(KID)

――そうですよね．「この言語を使え！」なんていっている人のいうことをいちいち聞いていたら，コンパイラの製品たくさん買わなくてはならなくなりますね．

★うちの学校には初代PC-8801があるんだ！

今回の付録のFDVer2.30はとてもよかった．また付録にきたいします!!うちの学校には初代PC-8801があります．しかもまだ動きます．でも，BASICバージョンが古すぎて円も書けません．時代の進歩を感じました．でもRAもあります．この差はいったい……ぼくはCユーザなので，BASICがわかりません．途中でBASICをやめてCに走ったからです．でも，それがいまはよかったのだと思います．(えーしゃくん)

――編集部で動いているいちばん古いマシンはEです．ディスプレイの色が変ですが，レーザプリンタにつながってちゃんと働いています．

★私の友人Aさん

私の友人「A」さんは，美しいものを見ると悲鳴をあげるが，Programmer's Pageを見て悲鳴をあげた．どうやら貴誌の構成に美しさを見いだしたようだ！(パワーボム)

――こういう直感的な反応というのは，なぜかうれしいものです．たくさん悲鳴をあげてください．

★改正不正競争防止法？

「改正不正競争防止法」が来月施行されます．ソフトウェアを開発する立場上，著作権に関する法律知識が，ますます十分に必要とされます（正当なソフトウェア開発を実施しているつもりでいるので，自分達の保護につながるけど……)．連載の「コンピュータの法律」はぜひ続けていただきたい．内容をもう少し開発する立場へ……(長谷川勝規)

――法律の問題は，立場によってニュアンスが変わってきますね．月刊プログラマーズ・ページですから，開発する立場に立つよう努力いたします．

★アイドル編が出ればいいな！

６月号の付録ディスク中の日本丸の壁紙は，他のプログラマにも非常に評判がよく，J-3300の中において立派にその役目を果たしている．ある先輩などは「アイドル編が出ればいいなぁ」などといっておりましたが，私個人としてはF40のような車がいいです．(MAは列島軍)

――日本で最初にBMPファイルをおまけにつけた(だからどうなんだといわれそうですが) 本誌としては，これからもできるだけ使っていただける壁紙を提供したいと考えております．アイドル編は，ライセンス問題があるからむずかしいでしょうが……．

★100MB以上が欲しいこのごろです．

パソコンを購入して１年がたった．Cの勉強とMASMを中心に，回路シミュレーションなどを，PC上で動かしてきた．いい加減にハードディスク100MB以上が欲しいこのごろです．MS-Cにしても，回路シミュレーションにしても，2HD×２だけでは速度面からも限界が見えてきたので，またWindowsにも興味がわいてきたし，秋頃には導入しようと計画しています．(寺嶋久雄)

――「100MBを導入しようと計画」できる状況がうらや

ましい．家で動いているのは，85msの20MBですもん．

★バージョンアップしてから早４か月

ろくすっぽモノにできないまま，TURBO-CからC＋＋にバージョンアップして早４か月．いまだ私のC＋＋は，ANSI-Cのソースしかコンパイルしたことがないという話である．おまけに，愛機のスロットには最近FM音源なるモノが差し込まれ，日夜トロイカやフリーソフトのゲームサウンドを奏でているという……．これでは，いかん，いかんと思いつつ，マウスを持つ手に力が入る．というワケで，くたびれてきたマウスを新調したい．プレゼントのマウスよ当たれ！！（安土誠二）

――あッ，こういうはがきを掲載してはいけないのかな？「ドラゴン・ポスト」にでて，マウスも当たる確率は低いような気がする．でも，多くのパソコンユーザーの典型的な姿が書かれているようでおもしろかったから……，ゴメン．

★私のHDは80MB

私のHDは80MB．２領域に分けまして，そのボリュームラベルが「Tool Box」と「Toy Box」．昔，はじめて自分専用の工具箱に工具を買い集めたときの，何ともいえぬうれしさと，机の引き出しの中に，ふる～いガラクタの秘密めいた楽しさを，HDの中に再現しようというわけです．なかなかいけますよ．FDとMIFESがあればシステムを組むのもラクラクです．自分専用のシステムを構築する楽しみは，こたえられませんネ，ウッシシ．（鵜木昌博）

――いいですね．ToolBoxとToyBoxなんて．HD01，HD02なんてボリュームラベルつけるよりずーと楽しそう．

★世界地図がお気に入りです！

私もLHA目当てで買ったくちです．NBCCでは英語版のWin3.0を用いていますが，最近，米国EMS社がディストリビュートするWin3.0用PDS集（260本36枚組）を買いましたが，おもしろいソフトやビットマップがあります．ウォールペーパーとして世界地図がお気に入りです．（高橋穣）

――BBSでそのPDSの話を読んでから，非常にうらやましく思っていました．BMPファイルがいっぱいはいっているそうですね．ぜひ見てみたい．

★ズルイ男です

ディスクが付属するようになって購入をはじめたズルイ男です．でもって，アンケートはがきをだしたのも，Pascalのプレゼントがあるため……うーん，なかなかセコいんだっ!!（坂本英久）

――そういう「ズルイ男」をあてにして，付録ディスクをつけた本誌も，なかなかセコいのでは……？

お詫び

本誌７月号の付録フロッピーディスクの制作に手違いがあり，表紙とディスクの袋に表示してありました「通信ソフトWTERM」が収録されていませんでした．ご迷惑をおかけした読者・書店のみなさまに深くお詫び申し上げます．

当面の対策として，本誌に添付の「愛読者カード」をお送りいただいた方には，「WTERM」を収録したディスクをお届けすることといたしました．また，書店様にご協力をいただき，「愛読者カード」によるディスクの請求をお願いするとともに，SE-NETによるお知らせもいたしましたが，まだ「WTERM」がお手元にないためにお困りの方は，７月号の「愛読者カード」により弊社までご請求いただきますようお願いいたします．折り返し「WTERM」を収録したディスクをお届けいたします．

なお，８月号（本号）の付録フロッピーディスクにも「WTERM」を収録いたしました．

弊社では，このような不手際を繰り返すことのないよう，十分注意して雑誌制作にあたる所存です．今後とも「月刊プログラマーズ・ページ」のご愛読をお願いいたします．

1991年７月

株式会社翔泳社
月刊プログラマーズ・ページ編集部

付録ディスクのご案内

このフロッピーディスクに含まれる実行ファイル，ソースリストおよびテキストファイルは「月刊Programmer's Page」1991年8月号に掲載されたものです．

付録のフロッピーディスクは，5"2HD（8セクタ／トラックMS-DOS）でフォーマットされています．3.5"や2DDなどへのコンバートサービスは行っておりません．なお，オンラインサービスSE-NET（03-5276-1297）でも，同じ内容のファイルを用意しています．

フロッピーディスクの内容は次の通りです．本誌およびディスクに含まれるドキュメントファイルをよく読んでからご利用ください．

■ディスクのおもな内容

●付録ディスクについての説明
ディレクトリ名：ルート
収録ファイル名：
README.BAT（説明を読むためのバッチファイル，PC-9801専用）
README.DOC（ディスクの説明）
ONLINE.BAT（ONLINE.DOCを読むためのバッチファイル，PC-9801専用）
ONLINE.DOC（オンラインソフト採集帳の説明）
V.EXE（ミニページャー，PC-9801専用）

●特集「はじめてのオブジェクト指向プログラミング」
①「オブジェクト指向概念の基礎」(p.2)
ディレクトリ名：TOKU
圧縮ファイル名：OBJ.LZH
③「C++の現状と未来」(p.23)
ディレクトリ名：TOKU
圧縮ファイル名：CPP.LZH

●プログラム『遊』工房 (p.62)
「サイクロイド／トロコイド／スピログラフ描画」プログラム
ディレクトリ名：PLAY
圧縮ファイル名：CYCLOID.LZH

●MS-DOSツール・ラボ (p.76)
FORTHコンパイラ「FAST」
ディレクトリ名：TOOL
圧縮ファイル名：FAST.LZH

●98活用アセンブラ技法 (p.88)
ディレクトリ名：98ASM
圧縮ファイル名：RESIDENT.LZH

●実践C塾 (p.101)
ディレクトリ名：JUKU
圧縮ファイル名：STRING.LZH

●綴じ込み付録
ディレクトリ名：DATA
収録ファイル名
SUMMARY.TXT

●オンラインソフト採集帳 (p.136)
本誌今月号のオンラインソフト採集帳で紹介したデバイスドライバ選択プログラム「CINIT.SYS」,「DCP」の新バージョン，7月号で手違いのため収録できなかった多機能通信ソフト「WTERM」，これらのファイルを復元するのに必要な，高圧縮復元書庫管理ソフト「LHA」を収録しました．LHA, WTERMおよびDCPの簡単な使用方法についてはこのディスクのルートディレクトリにあるONLINE.DOCをご覧ください．
ディレクトリ名：ONLINE
収録ファイル名
LHA212.EXE　WTNSC1.LZH
DCP324.LZH　MLP063.LZH
WT830.LZH　CINIT102.LZH
WTDOC303.LZH
注意：オンラインソフトはその性格上，該当月号が発売されるまでに新しいバージョンが発表されている可能性があることをご了承ください．またサポートは基本的にパソコン通信のみで行われておりますので，ご注意ください．

●クイズ・ランダムスクリーン (p.142)
クイズ解読ツール「PUZLE」
本誌掲載のクイズ・ランダムスクリーンの画像をディスプレイ上に再現し，マウスを使って並べ替えることにより，解答を得ることができるユーティリティです．対応機種は日本電気のPC-9801シリーズです，詳しい使い方はPUZLE.DOCを参照してください．
ディレクトリ名：OMAKE
収録ファイル名
PUZLE.EXE（解読ツール，PC-9801専用）
PUZLE.DOC（PUZLEの説明）
CONFIG.SYS
AUTOEXEC.BAT
RNDSCREN.LZH（画像データ・圧縮）
HIYASHI.LZH（7月号クイズ回答データ・圧縮）
HIYASHI.BMP（Windows 3.0用壁紙）

＊使用上の注意
本誌掲載プログラムおよびデータを利用するには，MS-DOSの基礎知識が必要です．また，このディスクにはMS-DOSのシステムは入っていません．別途MS-DOSが必要です．なお，ソースファイルのコンパイルなどには，別途コンパイラやユーティリティが必要となる場合があります．

■圧縮ファイルについて
ディスクスペースを有効に使うために，多くのファイルを圧縮した上で収録してあります．LZHという拡張子のついたファイルは圧縮ファイルです．圧縮ファイルは，高圧縮書庫管理ユーティリティLHAを使って復元してからご利用ください．LHAの利用方法については，ONLINE.DOCをご覧ください．

バーチャルスペースガイド

P子　某ソフトハウスで企画営業を担当している．26歳．
C平　某メーカーのシステムソフトウェア開発部門に所属．25歳．

C平　ビデオソフトのデータベースってないのかな？　タイトルも監督もわからない映画を記憶の断片だけで検索できると便利だと思うけど．

P子　そりゃー，むずかしいんでないの．91年版の「ビデオソフト完全カタログ」（角川書店）という本にはたしか1万2千近くのタイトルがあったような……．データベース化もいいと思うけど，移り変わりが激しいでしょ．とくに，C平の好きなAVなんかは．

C平　なにAVって．否定はしないけどそういうのは，店頭でみればいいわけで，名作みたいなのだけでもデータベースになっていればなと思うわけ．

P子　で，探したかった名作というのは？　キーワード検索するとすればどうなるの？

C平　「アニメ」「猫」の絞り込み検索で，そのリストを見ればわかると思ったんだけど，猫が星の世界を旅する話だったな．

P子　それって，もしかして「銀河鉄道の夜」じゃない．宮澤賢治の「銀河鉄道の夜」を知らなかったわけ！

C平　そ，そういえば，学校で習ったような……．でも，猫がでてくる話じゃなかったような……．

P子　あれはね，ますむらひろしの猫のキャラクタを使ってアニメ化しているの．ますむらさんは，昔「マンガ少年」という雑誌で宮澤賢治を意識した漫画を描いていたんだけれど，猫を使ったのがとてもうまいと思ったわ．

C平　よく知っているじゃない．

P子　だって，マンガ少女だったからね．10年くらい前は．

■銀河鉄道の夜（1982）

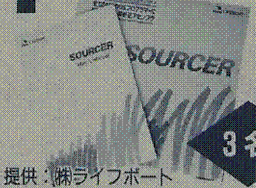
朝日新聞社・ポニーキャニオン
監督：杉井ギサブロー
声：田中真弓

宮澤賢治の「銀河鉄道の夜」のアニメ化．登場人物にジョバンニやカンパネラという名前を使ったのは，物語の仮想化を進めるための手段だったと考えることができるが，人間を主人公として視覚化した場合，変に教訓的になるなど違和感を生じかねない．その問題をうまく解決したのが，猫による視覚化であった．



読者プレゼント

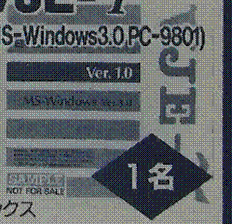
1 SOURCER



3名

提供：(株)ライフボート

2 VJE-γ (MS-Windows3.0,PC-9801)



1名

提供：(株)バックス

3 ボーランドジャパン特製カード型ラジオ



10名

提供：(株)ボーランドジャパン

4 日本語入力読本



3名

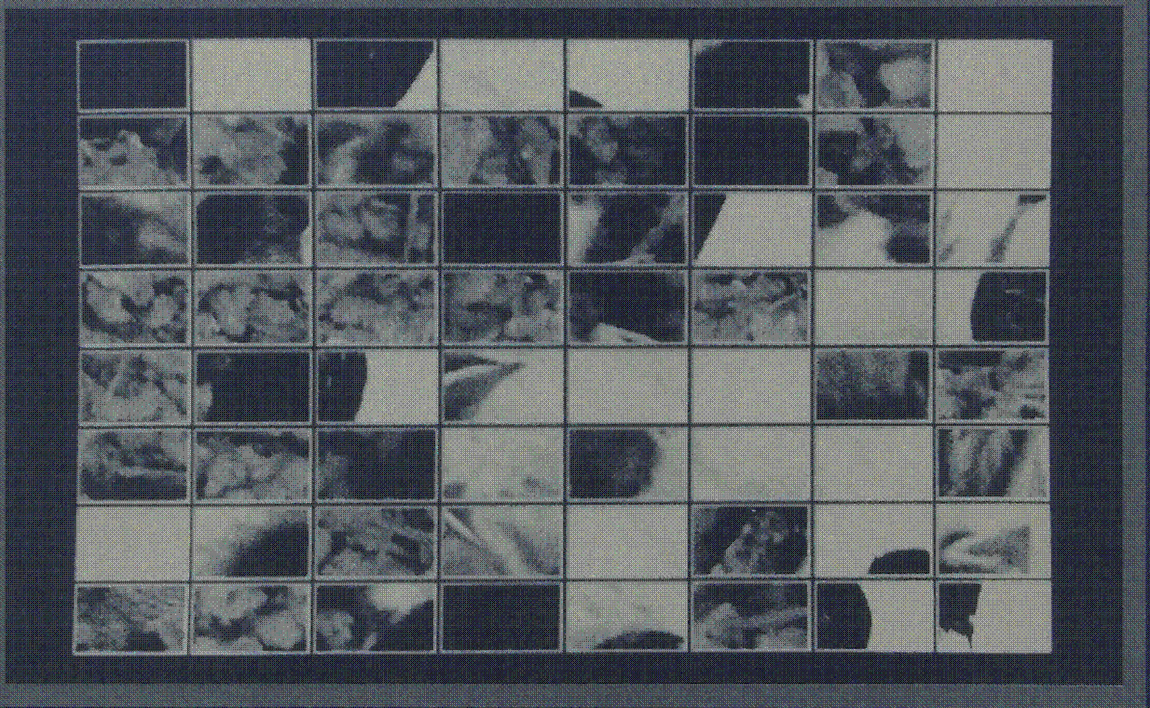
翔泳社・PC-PAGEシリーズ新刊

応募は本誌に添付のアンケートはがきをご利用ください．（1991年8月17日消印有効）．なお，応募者多数の場合は，抽選になります．当選者発表は発送をもって代えさせていただきます．

クイズ・ランダムスクリーン

ある１枚の画像を64分割し，ランダムに並べ替えてあります．
さて，もとの画像は何を写したものでしょうか．

ヒント：つばくろと呼ばれる渡り鳥．でも若い男の人をこう呼ぶことも…



正解者の中から30名様に本誌表紙イラスト（日暮修一画）入り月刊プログラマーズ・ページ特製オリジナルマグカップをさしあげます．

応募方法：本誌に添付のアンケートはがきをご利用ください．

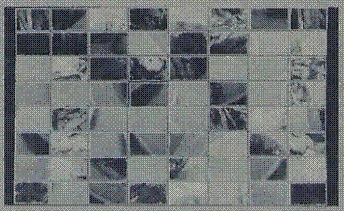
締め切り：1991年8月17日（消印有効）

なお，当選者の発表は発送をもって代えさせていただきます．

＊画像データは，付録ディスクやBBSでもサービスしています．

7月号の問題と正解

【問題】



【正解：冷し中華】



月刊プログラマーズ・ページ オンラインサービス（SE-NET）のお知らせ

付録にフロッピーディスクをつけたため，ディスクによるプログラムサービスは中止いたします．オンラインサービスのBBSは，多数の読者の登録をいただいておりますので，今後も継続いたします．編集部とのコミュニケーションの場としてご利用ください．

●BBSへの接続

SE-NET　03-5276-1297

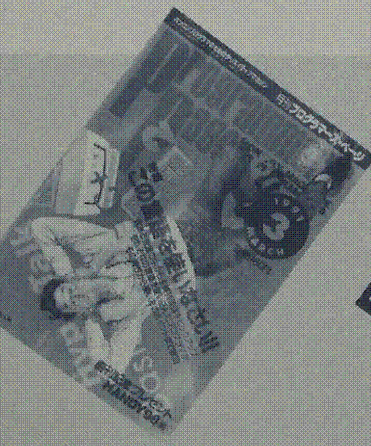
①接続するとIDの入力を求めてきますので，GUESTと入力しリターンキーを押してください．
②お知らせを表示後，メインメニューのコマンドプロンプトが表示されます．
③月刊プログラマーズ・ページへのご意見などは「ドラゴンポスト」ボードに書き込んでください．
④ファイルの転送が終了したら，0と入力して回線を切ります．

Programmer's Page
MONTHLY MAGAZINE FOR PROGRAMMERS

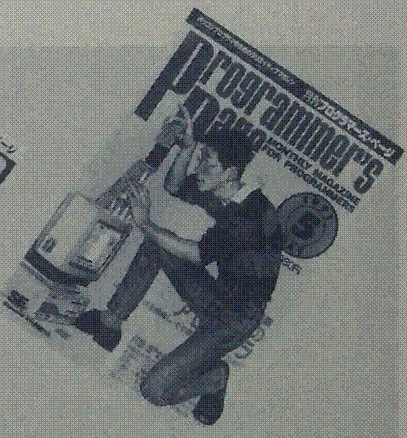
月刊プログラマーズ・ページ バックナンバーのご案内

■定価・各980円

（1991年１月号「特集・プログラマの仕事場」・２月号「特集・Windowsの時代である!」は品切れです）







1991年3月号

特集・この言語を使いなさい!!

数あるプログラミング言語の中から14をピックアップ，それぞれの言語を使い込んだプログラマが，その特性から使い方のポイントまでを解説する。

1991年４月号 5インチFD付き

特集・C読本

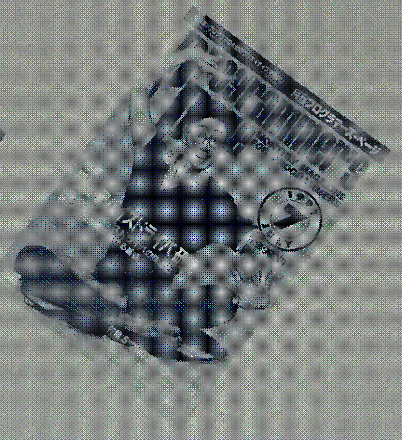
パソコンプログラミング言語の標準的存在になったＣ言語ととことん付き合うための大特集。入門からオブジェクト指向への流れまでを紹介する全５章。

1991年５月号 5インチFD付き

特集・基礎力としてのアセンブラ

プリミティブなプログラミング言語でありながら，それ故に無視することができないアセンブラ。コンピュータを理解し，その能力をひきだすためにもう一度アセンブラをみなおす。





1991年６月号 5インチFD付き

特集・研究「プログラミング統合環境」

今流行りのエディタ，コンパイラ，リンカ，デバッガを統合した開発環境について，その歴史的系譜から最新のシステムまで徹底的に検証する。

1991年7月号 5インチFD付き

特集・最新「デバイスドライバ」研究

パソコンの拡張性を支えるソフトウェアインターフェースであるデバイスドライバ作成の技法と特徴をMS-DOS，Windows，OS/2の各環境ごとに解説する。

「月刊プログラマーズ・ページ」のバックナンバーは，ご注文いただければ，全国どこの書店でもお求めになれます（書店に届くまで１か月ほどかかります）。
小社に直接ご注文なさるときは，何月号かを明記の上，現金書留にて雑誌価格＋送料を下記までお送りください。３月号から５月号の送料は１部260円です（６月号は１部81円，７月号は，１部86円）。２冊以上の場合は，号数・冊数により送料が異なりますので，小社までお問い合わせください。

〒102 東京都千代田区平河町2-4-14
平河町KSビル4階
（株）翔泳社
マーケティング部PP係宛
TEL 03-3263-0447

Programmer's Page

MONTHLY MAGAZINE FOR PROGRAMMERS

月刊プログラマーズ・ページ9月号

(8月18日発売)

付録: 5″ 2HDフロッピーディスク付き!!

特集
グラフィックスプログラミングの世界

コンピュータグラフィックスは，多くの人から注目を浴びる華麗な存在である．それを実現するためのグラフィックスプログラミングの基礎，フレームバッファの活用法，レイトレーシング，DOS/Vで注目を集めるIBM PCのグラフィックス，Windowsにおけるグラフィックスなど，プログラミング技法を徹底解説．

月刊プログラマーズ・ページ　1991年8月号
1991年8月1日発行

発行所　株式会社翔泳社
東京都千代田区平河町2-4-14平河町KSビル4F・5F
TEL 03-3263-0447　〒102
発行人　長廻健太郎
編集長　清水　隆
副編集長　鈴木嘉平
編集スタッフ　杉野貴美子＋上田純美礼＋芳山充弘
デザイン　SE（翔泳社デザイン研究所）
広告スタッフ　相澤正夫＋澤本良知＋種村嘉彦
協力　根本裕三郎＋(株)ドモン・マインズ＋佐藤裕美
＋小林弘人(図版デザイン)
写植　株式会社東巧社＋株式会社トーヨー・プリント社
印刷　凸版印刷株式会社
定価　980円（本体951円，税29円）

EDITOR'S PAGE

●「著者にパソコンネットの有名人が多い」とは，よく人からいわれることだ．読者からのはがきにもそういうご意見があったと思う．結果的にそうなっているが，やはりそれは“結果的”なのだ．人に紹介していただいた著者，講演を聞いて原稿をお願いした著者，ネットで知り合った著者，いろいろである．そうした，人たちがたまたまパソコンネット上でアクティブであるということになる．別々に知り合った人が意外なところでつながっていたり，別のネットで活躍していた人が新しく加入してきたり……それだけでも楽しい世界だ．(TA)

●DOS/Vを使い始めてずいぶんになるが，このDOS/V上での開発システムというのをまだ見たことがない．英語モードで動作する処理系を使えば一応開発はできるのだが，やはりDOS/Vにきちんと対応した処理系が欲しい．かといって，日本IBM純正というのでは値段がとんでもなくなりそうだし，やはりここはサンデープログラマの強い味方，ボーランドジャパンにDOS/V用のターボシリーズを出してもらうというのが正解だろう．ボーランドジャパンさん，やってくれませんかねぇ……．と思っていたらMS-Cが出た／（嘉平）

●ある日の午後，ついにわが家にMacintoshが来た．マニュアル片手にあっという間にインストール完了，いざ使おうと思ったら，ソフトが何もないことに気づく．しかたなくうちの猫たちの声を録音して遊んだ（結構むなしかったけど，これだけでもまだましかナ?)．「パソコンはソフトがなければただの箱」を実感した午後だった．しかし，かさばって困るのはパソコンの空箱．下取りや引越のことを考えれば，とっておいたほうがいいと思うのですが，皆さんはどうしていますか？（ま）

●先日，書店である雑誌を見ていたら，投稿欄に(23歳　プログラマ）とあるのがふと目にとまった．読んでみると投稿者は女性だった．読者ハガキによれば読者の99パーセントは男性．でも，世にプログラマを名乗る女性は確実に増えているらしい．女性プログラマのみなさまもぜひハガキを送ってください．それにはもっと実用的な読者プレゼントを考えなくちゃだめかしら……．(kos)

●5月末にスズキが「GSX-250S KATANA」を発売した．最新バイクと同じ水冷の心臓を持ちながら，ちょうど10年前にケルンショーでデビューした「GSX-1100S 刀」のプロトタイプと同じハンス・ムートのデザインに身を包み僕の前に表れた．毎日，僕の頭の中では資金繰りの計画が着々と進んでいる．本誌が出る頃には乗っていると信じたいものである．(Yoshi)

お詫びと訂正　本誌7月号「IC千夜一夜」(75ページ)の図1DTLの回路図と図2TTLの回路図が入れかわっておりました．訂正するとともにお詫びいたします．

愛読者カードご記入のお願い

本誌をお買い上げいただき、誠にありがとうございます。
お手数ですが、今後の誌面づくりの参考にさせていただきたいと存じますので、
各欄にご記入の上、ご返送くださるようお願い申しあげます。

月刊プログラマーズ・ページ READER'S SURVERY CARD 愛読者カード

本誌の号数　　年　　月号

本誌を知った理由
□1.書店店頭　□2.他社出版物（　　　）
□3.小社出版物　□4.DM　□5.口コミ

本誌を購入した書店　　市（区）　　書店

パソコンの所有形態（機種名）
□1.個人所有　（　　　）
□2.職場にある（　　　）
□3.持っていない

よく使用するソフトウェア

よく使用するOS

今月号でおもしろかった記事（いくつでも）

今後取り上げてもらいたい特集のテーマ

貴方が選ぶベスト3

パソコン雑誌	①	②	③
ソフトウェア	①	②	③
パソコン本体	①	②	③

本誌についてのご感想、お気づきの点、その他どのようなことでも結構ですので、ご意見をお聞かせください（「ドラゴン・ポスト＝読者の声」欄に掲載させていただく場合もあります。ペンネームをご希望の方はあわせてご記入ください）。

月刊プログラマーズ・ページ 広告資料請求カード

8	会社（学校）名		9	所属（学部）		10	従業員数	

※下記の該当する番号をご記入ください。

11	業種		12	職種		13	パソコン利用状況	

業種　①製造業　②金融・保険・不動産業　③商社・卸・小売業
④建設業　⑤通信・運輸　⑥システム・ソフト
⑦その他サービス業　⑧公務員　⑨その他

職種　①コンピュータ（ハード）技術者　②コンピュータ（ソフト）技術者　③エレクトロニクス技術者
④プログラマ　⑤その他の技術者　⑥専門職
⑦事務・営業職　⑧経営管理職　⑨商工サービス
⑩教職　⑪学生　⑫その他

パソコン利用状況　①会社で利用　②個人で利用　③会社でも個人でも利用　④利用していない

●希望される広告資料　　年　　月号

希望項目 ＼ 広告資料請求No.												
A 購入希望	A	A	A	A	A	A	A	A	A	A	A	A
B 説明希望	B	B	B	B	B	B	B	B	B	B	B	B
C 資料希望	C	C	C	C	C	C	C	C	C	C	C	C

定期購読のご案内

綴じ込みの振替用紙を使って、本誌の定期購読をお申し込みになれます。
1年間12冊＝11,760円（本体11,417円/税343円）
郵送料サービス

月刊プログラマーズ・ページ READER'S SURVEY CARD 愛読者カード

氏名

性別 □1男 □2女

生年月日 19 年 月 日

住所 〒

電話 （ ） －

職業
□1ソフト技術者 □2ハード技術者 □3技術系会社員
□4事務系会社員 □5営業系会社員 □6教師
□7一般公務員 □8経営管理職 □9理系学生
□10文系学生 □11その他（ ）

ユーザー区分
□1プロ
□2マニア
□3一般ユーザー

プレゼント記入欄
希望商品番号

クイズ・ランダムスクリーン回答欄
答

＊当社で記入
1 SE
2 受付No.

月刊プログラマーズ・ページ広告資料請求カード

3 会員番号
＊すでに会員番号をお持ちの方はご記入ください。

4 フリガナ 姓 名
氏名

5 生年月日 19 年 月 日

6 自宅住所 〒 電話 （ ）

7 会社住所 〒 電話 （ ）

○資料送付希望先の番号に印をつけてください。

払込票

通常払込料金加入者負担

口座番号	東京 4-15493
加入者名	株式会社 翔泳社
金額	11760
払込人住所氏名	※
備考	
受付局日附印	

切り取らないで郵便局にお出しください。

記載事項を訂正した場合は、その箇所に訂正印を押してください。

払込通知票

通常払込料金加入者負担

口座番号	東京 4-15493
加入者名	株式会社 翔泳社
金額	11760
料金	払込み　特殊　円
備考	
払込人住所氏名	※（郵便番号　　） TEL（　　）
受付局日附印	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。（郵政省）

各票の※印欄は、払込人において記載してください。

・画像・自動・マイクロソフト
ェアとハードウェアの企画・開
在は、制御系で獲得したノウハ
らに新しいフィールドへチャレ

事よりも、そこで活動している
ることによって、リアリティの
う趣旨で、ソフトビジョンの中
グラマである関氏に取材した。
をこなすのではなく、自分の興
ように仕向けてきた関氏は、ノ
ラマとは一線を画した、創造性
している社内報に、エッセイや
に造詣の深い人、という関氏の
1時間、会議を終えたばかりの

すことはちょっと」といいながら、デモンストレーションの画面を表示してくれた。

計測関係か制御関係のアプリケーションだというが、表示された画面は、ゲームソフトかWINDOWSなどと見間違うほど、きれいで凝った画面だった。

つい頼まれたこと以上の仕事をしてしまう関氏。画面に色をつけたり、使いやすさを求めてインターフェースに凝ったり、およそ会社から見れば、無駄な努力だと思われることを、短期間

るつもりだったという。しかし、本業のサラリーマンでは、思うようなパソコンを入手することが困難になり、25歳でソフトハウスに転職した。

最初の会社では、ゲーム関係のプログラム開発に従事して、インテルのZ80とアセンブラを駆使したが、そのうちパソコンの限界と重荷を感じるようになり、もっとパワフルで使いやすいものを探したという。

そんな関氏が選んだOSとCPUが、OS-9にMC68000だった。

OS-9の魅力

「非常にマイナーですよ」と最初には、OS-9に対する率直な印象をた。思わず「野球はどこのファンか？」と尋ねると、「大洋です」とが返ってきた。「とくに熱心な野球ンではないが、大洋が最下位になときに、ファンになろうと思った」という。

OS-9に取りつかれたのは、やはりすっきりしたアーキテクチャを求めてのことだといえる。「86系の石を基準にしたMS-DOSのメモリ管理は、セグメント単位で繁雑な上に、最近ではEMSやXMSのように、建増しのようなメモリ増設が行われている。その上に、UMBのように地下にまで潜るような技術も駆使してきたが、そんな無駄な労力をそそぎ込むより、それをもっと違うことにつぎ込むべきだ」という。

定期購読のご案内

■定期購読は、1年間12冊と限定させていただきます。
■購読料のお支払は、この振替用紙をご利用ください。銀行振込、現金書留によるご送金は、ご遠慮ください。
■振替用紙に、ご住所、お名前をご記入のうえ、郵便局の貯金窓口にてお支払ください。
■小社より領収書の発行はいたしませんので、郵便局にて「郵便振替払込金受領証」をお受取ください。
■お申し込みいただいた方には、弊社より直接発送いたしますので、到着が発売日より遅れることがございますが、ご了承ください。

月刊プログラマーズ・ページ　定期購読直接申し込み書

この欄は、加入者あての通信にお使いください。

Programmer's Page MONTHLY MAGAZINE FOR PROGRAMMERS

年　　月号より

年間購読料 11,760 円(税・送料込み)

■ご希望の購読開始月号をご記入ください。

■送付先（住所・氏名は正確にご記入ください）

住所	□□□-□□	フリガナ	
		氏名	
		TEL	

この払込通知票は、機械で使用しますので、下部の欄を汚さないよう特に御注意ください。また、本票を折り曲げたりしないでください。　（郵　政　省）

郵便はがき

102-00

東京都千代田区平河町2-4-14
平河町KSビル

株式会社 翔泳社

月刊プログラマーズ・ページ
広告資料請求係行

2	受付No.	

マーズ・ページ広告資料請求カード

＊すでに会員番号をお持ちの方はご記入ください。

名	5	生年月日	19　年　月　日

綴じ込みの

□10文系学生　□11その他(　　　　)　ユーザー

プレゼント記入欄	クイズ・ランダムスクリーン回答欄
希望商品番号	答

6	自宅住所	〒　電話　(　　)	資料送付希望先の番号に○印をつけてください。
7	会社住所	〒　電話　(　　)	

ちょっと気になるプログラマ

株式会社ソフトビジョン

可能性のある人や
プロジェクトへの投資は惜しまない。

ソフトビジョンは、通信・画像・自動・マイクロソフト制御分野を中心にソフトウェアとハードウェアの企画・開発を行っている会社だ。現在は、制御系で獲得したノウハウと経験を基礎として、さらに新しいフィールドへチャレンジしようとしている。

今回は、会社案内的な記事よりも、そこで活動している人とプロジェクトを紹介することによって、リアリティのある現場の姿を伝えるという趣旨で、ソフトビジョンの中でも異彩を放っているプログラマである関氏に取材した。

会社から与えられた仕事をこなすのではなく、自分の興味のある仕事を推進できるように仕向けてきた関氏は、ノルマをこなすだけのプログラマとは一線を画した、創造性のある人物だ。月刊で配布している社内報に、エッセイやコラムを書き、音楽や映画に造詣の深い人、という関氏の事前情報を得て、待つこと1時間、会議を終えたばかりの氏をたずねた。

OS-9にのめりこんで

関氏の仕事場は、10台近いマシンに囲まれた一角にある。そこにあるマシンは、すべてMC68000系のCPUを搭載した機種ばかりだ。これは、関氏の社内努力の結果だという。

「現在開発しているアプリケーションの内容は、企業秘密なので詳しく話すことはちょっと」といいながら、デモンストレーションの画面を表示してくれた。

計測関係か制御関係のアプリケーションだというが、表示された画面は、ゲームソフトかWINDOWSなどと見間違うほど、きれいで凝った画面だった。

つい頼まれたこと以上の仕事をしてしまう関氏。画面に色をつけたり、使いやすさを求めてインターフェースに凝ったり、およそ会社から見れば、無駄な努力だと思われることを、短期間に手早くこなすのだ。そして、その無駄な努力を有益なノウハウに変えてしまう魔術が、関氏には秘められている。

関氏はどのようにして、OS－9/68000系にのめりこんでいったのだろうか。
その経緯を語ってもらった。

「最初は普通のサラリーマンだったんです」と関氏は、ソフトビジョンに入社する以前の経歴に触れる。

パソコンは趣味にとどめておいて、本業で稼いだお金をパソコンに投資するつもりだったという。しかし、本業のサラリーマンでは、思うようなパソコンを入手することが困難になり、25歳でソフトハウスに転職した。

最初の会社では、ゲーム関係のプログラム開発に従事して、インテルのZ80とアセンブラを駆使したが、そのうちパソコンの限界と重荷を感じるようになり、もっとパワフルで使いやすいものを探したという。

そんな関氏が選んだOSとCPUが、OS－9にMC68000だった。

OS－9の魅力

「非常にマイナーですよ」と最初に関氏は、OS－9に対する率直な印象を述べた。思わず「野球はどこのファンですか？」と尋ねると、「大洋です」と答えが返ってきた。「とくに熱心な野球ファンではないが、大洋が最下位になったときに、ファンになろうと思った」という。

OS－9に取りつかれたのは、やはりすっきりしたアーキテクチャを求めてのことだといえる。「86系の石を基準にしたMS－DOSのメモリ管理は、セグメント単位で繁雑な上に、最近ではEMSやXMSのように、建増しのようなメモリ増設が行われている。その上に、UMBのように地下にまで潜るような技術も駆使してきたが、そんな無駄な労力をそそぎ込むより、それをもっと違うことにつぎ込むべきだ」という。



ちょっと気になるプログラマ

SOFT VISION　関　勝成

「好きなものは？」という質問に対して、「ホープ軒のラーメン」と答えてくれた。「奥様は？」と尋ねたら「あれは別格」と笑って答える愛妻家だ。昭和34年生まれの天秤座

とくにVMEバスやSCSIインターフェースを利用する制御やマルチタスクでは、UNIXよりもカーネルが小さくて、ROM化できるOS-9が優れたOSだというわけだ。

パソコン通信のマニアでもある関氏がよくアクセスしているのは、Nifty―SERVEのOS-9フォーラムだという。

新製品やニュースも乏しく、あまり盛り上がりのない、風前の灯のような会議だが、OS-9のプロばかり集まっているそうだ。ちなみに、OS-9フォーラムへはGO FOS9で移動できる(ちなみに関氏のIDは、MHC01210)。OS-9に興味があれば、一度Niftyにアクセスしてみてはいかがだろうか。

社内個人努力

冒頭でも触れているが、関氏は、一般的なソフトハウスの経営者から見れば、採算重視のプログラマではない。自分の好きな仕事ばかり選んで、挙げ句の果てに、顧客に対して開発側の意見を通してしまう。

その結果が、社長までも説得して、OS-9プロジェクトの推進役に納まることになってしまった。通常ならば起こり得ないことだが、そんなわがままが通ってしまう原因は、ソフトビジョンの社風にある。

お定まりの社風がないソフトビジョンだが、ソフトハウスとして生き残るためには、受託専門の会社ではなく、提案型の企業になる必要があると考えられている。そのため、少しでも可能性のある個人やプロジェクトには、積極的に投資して、個人の資質を伸ばす環境を用意している。

もちろん、誰もが可能性を持っているわけだが、それを現実化するためには、もうひとつの個人努力も必要になる。　それが、関氏の行ってきた努力の結果だ。

OS-9のプロジェクトは、一種の賭けともいえるソフトビジョンの挑戦だ。

危険を承知のプロジェクトといえる。そんな重要な決断を、会社の中で通すために、関氏自身は、「自分に自信を持つこと」の大切さを訴える。

「つまらない仕事ばかりしていると、大坂弁変換フィルタのようなPDSを作ってしまう」と昔を振り返る。いまは、OS-9がおもしろくて、PDSを作って遊んでいる暇がないという。

関氏は、若いプログラマに向けて3つのアドバイスをくれた。

第1のアドバイスは、「TSRに凝るなら、最初からマルチタスクにしなさい」だ。遊びで作るならTSRもいいが、仕事に使うならば、もっと楽ができるOSを使うべきだという。

そして「LANを構築するなら、マルチユーザーを意識させないものでなければだめだ」という。

最後に「OS-9を知りたい人は、いつでもここにきて触ってみて欲しい」と結んだ。

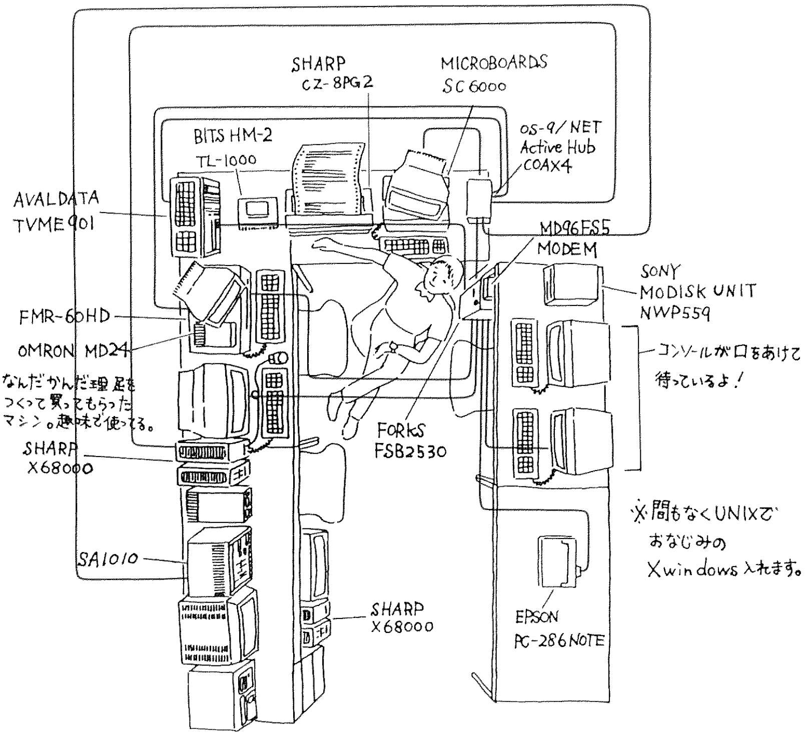
ソフトビジョン内で、関氏と一緒にプロジェクトを推進する仲間は、現在は約8人いる。若いプログラマたちにとって、OS-9は、「まだまだわけがわからないOSだろう」と関氏は笑う。

それでも、なるべく早いうちに、自分と同じレベルに達して欲しいと願っている。もちろん、そのときには、関氏自身も、「もっとずっと先を歩いていたい」と笑いながら気を引き締めていた。

●

関氏のいうようにOS-9は「マイナー」な存在だ。しかし、技術者としてそれを選択した関氏が、どのようにして「プロの仕事」見せてくれるのか？今後の活躍に期待したい。

拘(こだわ)りの美学



上記のイラストの主が関勝成さん(31歳)。　そしてずらりと並んだマシンの山は、彼の良き相棒たち。「社内に於ける時空間、および共有物資の私物化、これこそが自由創造の基本である」と関さん。　先行投資の重要性を、身をもって訴え証明してきた努力の人である。「仕事は遊び」と言いきるその姿勢には、遊びも仕事も徹底的に、それも人の指図を受けずにやらないと気がすまないという、拘りの美学が見える。　電話で社のシステムに侵入、いやログインして深夜自宅からソースファイルをつつくなんて事は朝飯前、ついには電話料金節約のために、ビルの屋上にアンテナを立て「これからはパケット通信の時代だ」と吠えながら、空中にエラーメッセージをふりまくというパフォーマンスまで見せたという。　突然夜中にラーメンを食べるためだけに千駄ヶ谷までクルマをとばし、ああうまかったと言いながら帰って来て、再びキーボードに向かっていたかと思えば、VMEのマニュアルを枕に、ガーガー寝ていたりする。　そんな彼の左手には、文字盤のない腕時計が静かに時を刻んでいる。

会社プロフィール

●昭和59年4月2日　代表者●代表取締役　鈴木和芳　資本金●990万円　従業員数●124名(男:90名、女:34名)　平均年齢●25.5歳　売上高●7億4000万円(平成2年度実
内容●①FA＆ロボットなどの自動制御　②データ通信＆自動車電話などの通信制御　③TV局向け＆医療向けの映像制御・画像制御　④オンライン端末＆無線端末な
クロソフト制御　上記４分野を中心としたソフトウェア・ハードウェアの企画、開発、販売、コンサルティング　主要機種●HP64000、HP9000LAN、PC9801、他
言語●C、アセンブラ、他　主要取引先●日本電気㈱、日本電気無線電子㈱、日本無線㈱、富士通機電㈱、松下システムエンジニアリング㈱、松下通信工業㈱、京セ
エンジニアリング㈱、東芝エンジニアリング㈱、昭和電子㈱

募集要項

●コンピュータエンジニア　資格●学歴不問　26歳位迄　※学生アルバイト及び新卒者大歓迎　給与●月給　固定給制19万円以上　※経験・年齢・前職を考慮の上、相
決定します。26歳支給例／31万6500円　アルバイト／時給1000円以上(時間相談に応じます。)　待遇●昇給年１回(3年度実績12%)、賞与年２回(2年度実績7ヵ月)、通
(2万4000円迄支給)、各種社会保険完備、住宅・家族・精勤・時間外・役職手当、退職金制度、財形貯蓄制度、社内旅行（3年10月グアム予定)、クラブ活動（野球部、テニ
ケット部、ウインドサーフィン部　他多数クラブ　※会社から援助金支給)、ラップトップ・レンタル制度（長期間無料）　勤務地●本社（相模原）　勤務時間●フレ
イム制(コアタイム10:30〜15:30)　※タイムカードはありません。1日標準労働時間８時間　休日休暇●完全週休２日制（土・日)、祝日、夏期・年末年始・ＧＷ・有給（
日から)・慶弔休暇　※年間125日＋有給　応募●電話連絡の上、履歴書(写真貼付)・職務経歴書(職歴のない方は自己PR書)を持参ください。郵送も可。　※FAX応募
ます。※応募の秘密厳守します。※入社日は相談に応じます。※応募の問い合わせ歓迎します。※大学生の会社訪問歓迎します。　交通●ＪＲ横浜線相模原駅徒歩５

採用関係連絡先

9　神奈川県相模原4-3-14相模原第一生命ビル「株式会社ソフトビジョン　人事部人事課」　フリーダイヤル0120-211744　採用担当／海老沢、斉藤

SOFT VISION

株式会社 ソフトビジョン

川辺義宏クンは、数理技研をどう考えているか。

第二話 ネットワークと個性に唸〜っの巻

「数理アカデミア」での僕の試練も無事終わり、
UNIX関連のネットワークソフトの開発を担当することになった。
ウーン、なかなか個性的な面々の集うプロジェクト・チームである。
「大丈夫、君も充分個性的だよ」との先輩の言葉に、僕は再び、ウーンと考え込んだ。
個性的なのは人に限ったことではない。仕事にも独自のポリシーがある。
とにかく、新しいことに挑戦するのが好きな会社である。
宇宙システム、ハイビジョン、統合化UNIX、X端末、NFSなど、
話題性に富んだ開発がいたるところで行われている。
既存の資料というものがほとんどないため、開発では調査・分析から始めなければならない。
これがシンドイところでもあり、面白いところでもある。
こうして、ウーン、ウーンと唸りながら開発に取り組む日々が始まった。
ドライバがらみの部分が多かったので、C言語だけではなく、
アセンブラもかなり勉強した。
また某メーカーの新規マシンだったため、
ハードウエアがらみの問題も発生し、悩まされることが多かった。
何か問題が起こる度に、あーでもない、こーでもないと、
皆で議論しているうちに、窓から見える新宿の夜の街は、
いっそう輝きを増していくのであった。

（次号に続く）

㈱数理技研は、今回大々的に、
強力なスタッフを多数募集します。
これは、そのための、
人材募集シリーズ広告・第2弾です。
興味をもたれた方の
ご応募をお待ちしています。



DATA

■募集要項

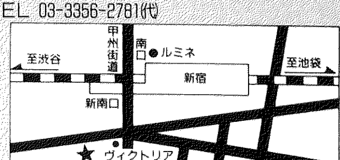
職種●システムエンジニア
資格●大卒以上もしくは大卒程度の学力を有する28才位までの男女。
※やる気・好奇心があり、技術を吸収する力がある方。
給与●経験・年齢を考慮の上、当社規定により優遇します。('91年新大卒基本給18万3千円)
待遇●昇給年1回('91年度実績平均8.0%)、賞与年2回('90年度実績平均7.2カ月)、各種社会保険完備、交通費全額支給、諸手当(子供・役職)、制度(住宅購入資金融資・退職金)、施設(独身寮・リゾートマンション)
休日休暇●完全週休2日制(土日)、祝日、年末年始、夏期休暇
勤務地●東京および山梨('92年4月開設)
勤務時間●フレックスタイム制
教育制度●数理アカデミアと称する体系的な理論学習の講座制度あり。技術セミナーへの参加や論文発表の機会多数。
応募●履歴書をご郵送下さい。追って詳細をご連絡します。
交通●JR新宿駅南口下車徒歩2分

■会社概要

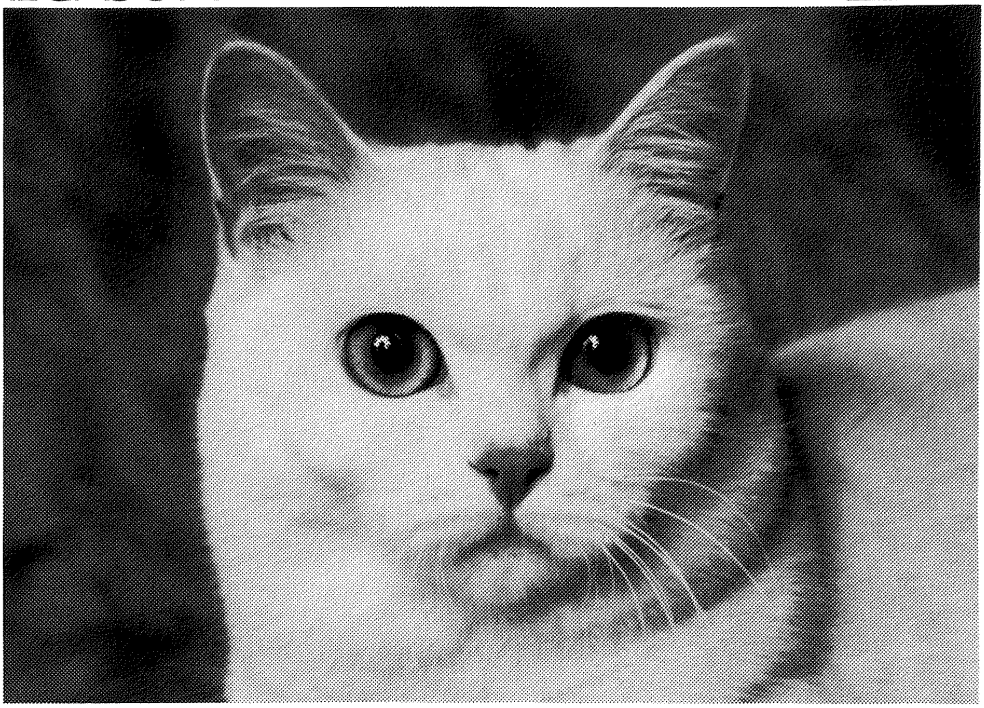
設立●1979年3月
資本金●4800万円
売上高●11億円('90年8月期)、13億円('91年8月期見込み)
社員数●60名(男45名、女15名)
事業内容●①UNIXはじめOS・通信系基本ソフトの開発
②大規模・分散処理システムの開発と同インテグレーション
③上記①、②の研究開発
④宇宙開発・核融合・数値解析など、科学技術計算の先端分野での研究開発
特徴●企業というより技術研究機関
促成栽培を否定した技術者教育
開発環境の完備
タイムカードなし、服装自由など自主性尊重
事業所●本社:東京都新宿
研究所:山梨県竜王
※'92年4月、山梨県竜王町のソフトウェアパーク内に山梨研究開発センターを開設

主要取引先●宇宙科学研究所、日本原子力研究所、NTT、NTTデータ通信、地球科学総合研究所、日本IBM、富士通、富士通研究所、日本電気、本田技研、日産自動車、IHI、住友電工、日立製作所、東芝、ソニー、アンリツ、日本信号、日本無線、ファナック、ヤマハ、NKKなど
技術提携先●UNIXインターナショナル、中国科学院軟件研究所、MENTAT Inc.(米)

株式会社 数理技研
〒160 東京都新宿区新宿4-1-9 新宿ユースビル
TEL 03-3356-2781㈹



MEGASOFT



ネコの手も借りたい!!

だから、人の手ならもっと、もっと大歓迎!!

■職種・条件

いずれも高卒以上、18～30歳程度の男女。

●ソフトウェア研究開発技術者

メガソフトのオリジナルパッケージソフトウェアおよび関連ハードウェアの開発担当。

→画像、音声、データを統合するマルチメディア通信、パソコンおよびワークステーション用次世代OS、LAN対応データベースシステムなどに経験・関心のある方。

●ハードウェア開発技術者

メガソフトオリジナルハードウェアの開発を担当。

→STARFAX用FAXモデムボードやISDN、音声メール用ボードなど、ソフトウェアとのバランスのとれた強力でエレガントな回路設計のできる方を求めます。

●カスタマサポート

電話でのカスタマーサポート、ショー、展示会でのプレゼンテーション等を担当。

→パソコン、ワープロを使える方歓迎。未経験者には実地指導します。

●営業スタッフ

メガソフト製品を広く多くの方に使っていただくための販売促進活動を担当。

→ユーザー、ディーラー向け販売促進活動およびパソコンメーカー向けOEM営業等が主業務です。バイタリティーのある方を希望します。

●企画スタッフ

オリジナルパッケージの企画・立案。製品化推進、広報等を担当。

→市場が求める製品をいち早くクリエイトし、ナンバーワン製品にまで仕上げる仕事です。積極的で創造性豊かな方を求めます。

■待遇

当社規定により優遇(経験・能力を考慮)。昇給年1回、賞与年2回(別に決算賞与:昨年度実績9ヵ月)。諸手当、交通費全額支給。各種社会保険完備。退職金制度。完全週休2日制(年間休日118日)。フレックスタイム制(開発部門)。社員旅行年2回。海外研修制度。

■勤務地

●大阪　大阪市営地下鉄御堂筋線「江坂」駅下車1分

●希望により東京勤務も可能です。

■応募

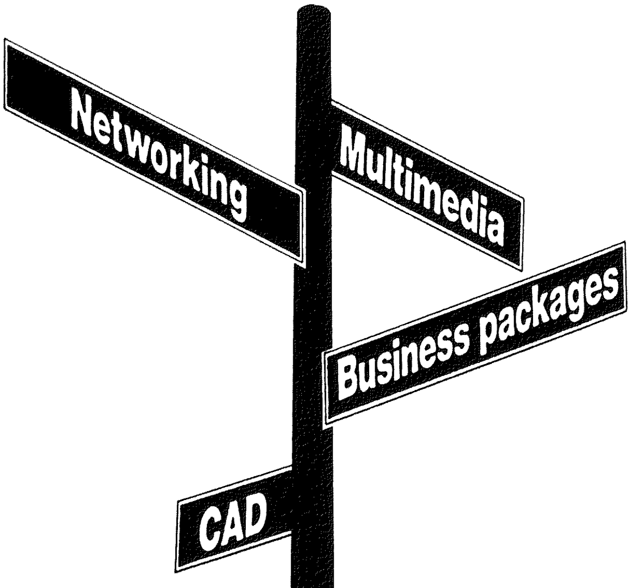
履歴書(写真貼付、希望職種明記)および職務経歴書(書式自由)を郵送してください。後日詳細をご連絡します。

■会社概要

資本金	1,000万円
年商	10億円(1990年8月期)
社員数	16名
業務内容	パソコン向け各種ソフトウェアの開発・販売
主要納入先	日本電気、日本IBM、シャープ、三菱電機、日立製作所、沖電気、ソフトバンク、ソフトウェアジャパン等(敬称略)
主要製品	パソコンFAXアダプター…………「STARFAX」 高性能スクリーンエディター……………「MIFES」 高速リモートディスクシステム………「MAXLINK」 仮想EMSドライバ…「MEMORY-PRO386」など

メガソフト株式会社

〒564 大阪府吹田市江の木町1-38 西谷東急ビル TEL06-386-2058(代) FAX06-386-4697



高水準の技術開発には
パーフェクト精神が根幹です。

ワンランク上の提案型ソフトウェアハウス＝エルテックスは、ユーザーや上司の求めている以上の開発を追求していく、我がままプロ集団の組織。常に高レベルの開発を行っていくためには、たえず外部の開発にアンテナを張りめぐらしておかなくてはなりません。その一環として、毎年数名、ラスベガスのコムデックスショー等、1週間程度の視察旅行に参加します。パーフェクトなものを求め続けていく技術者たちに、私たちは、たえず評価を与えていきます。

技術者随時募集

会社概要

ELTEX

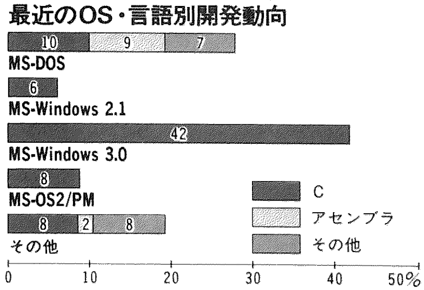
設　立	昭和60年12月14日
資本金	1,000万円
事業内容	パソコンを中心としたシステムの提案・開発・販売　①CD-ROM・LD など新しいメディアに対応したシステムの開発 ②ネットワークの端末システムの開発 ③特定業種向けCADの開発・販売 ④基本ソフトの開発
特　徴	大手クライアント向けのシステム開発が多く、ネットワーク関連システムやマルチメディアなど幅広く対応している。また、ＣＡＤを中心とした自社プロダクツにも積極的で、その比率も増加している。基本ソフトの分野では、ハードウェア専門のベンチャー企業とジョイントし、ＯＥＭ製品のＢＩＯＳ・ユーティリティなどを企画から手掛けている。
取引先	三菱電機、三菱自動車、日本航空　他
関連会社	株式会社ソーテック、学校法人福井情報ビジネス総合学院

募集要項

ELTEX

職　種	①システムエンジニア　②プログラマー
資　格	専門卒以上　20〜36歳の方　※Ｃ言語・アセンブラ言語経験者大歓迎　※未経験者でも好奇心・向学心の旺盛な方は歓迎します。
給　与	17万円（専門卒初任給）〜46万円（36歳課長、諸手当含む） 24歳例：23万5000円（情報処理１種手当・住宅手当含む） 30歳例：31万6000円（中級SE、情報処理１種・住宅手当含む）
手　当	時間外手当、役職手当（5000円〜６万円）、住宅手当（１万5000円）、資格手当（２種：5000〜特種：２万円）、通勤手当
賞　与	賞与年２回（平成１年度実績年間５ヵ月）、決算賞与年１回（平成１年度実績１ヵ月）
勤務地	本社（横浜駅近く） ◇派遣・出向はありません。
勤務時間	フレックスタイム制（コアタイム10:00〜16:00）試用期間中は標準時間9:00〜17:45
休日休暇	完全週休２日制（土・日）、祝日、年３回連休（GW・夏期・年末年始）、有給・特別・休暇
福利厚生	各種社会保険完備、退職金制度、財形貯蓄、社員旅行、海外研修、定期健康診断、社員寮、スポーツクラブ法人会員
応募方法	電話連絡の上、履歴書(写真貼付)、職務経歴書を持参し来社ください。郵送も受け付けます。
連絡先	〒220 横浜市西区南幸二丁目20番５号　東伸24ビル５階　℡(045)322-9667　総務課

最近のOS・言語別開発動向



採用HOT・LINE
TEL.(045)322-9667

株式会社エルテックス

募集要項

職　種	プログラマー及びマイコン回路設計者
給　与	基本給　月給19万円以上＋諸手当　※経験・年齢・能力を考慮の上、当社規定により決定します。
待　遇	昇給年１回、賞与年２回(昨年度実績6.5ヵ月)、交通費全額支給、各種社会保険完備、手当／役職・休日・残業・住宅（４万円支給）・資格(情報処理１種：３万円、２種：１万円)、独身寮(冷暖房付きワンルーム)、社内旅行、クラブ活動（BIG BOX法人会員野球部)、保養所
休日休暇	完全週休２日制(土・日)、祝日、年末年始・夏期・有給・慶弔休暇
勤務地	東京（高田馬場） ※自社開発につき出向・派遣はなし
勤務時間	9：00～17：30 (実働７時間30分)
応　募	電話連絡の上、履歴書(写真貼付) 持参ください。 ※入社日相談に応じます。 ※応募以外のことでもお気軽にお電話ください。
交　通	ＪＲ・西武新宿線高田馬場駅徒歩１分

会社概要

設　立	昭和58年４月１日
資本金	400万円
売上高	５億円（２年度実績）
従業員数	15名（男：11名、女：４名）
使用言語	アセンブラ、C
使用機種	UNIX、MS-DOS、PC-98、他
事業内容	ホテル・流通産業を中心としたシステム開発・製造

ミドランテック株式会社

東京都新宿区高田馬場1-34-6八光ビル２F――〒169

TEL03-3205-3503

担当――西沢

コンピュータを覚えた怪物たちが、私たちの宝物。



社内は非常にごちゃごちゃしている。ソフトと一緒にハードも独自で開発しているということもあるが、社員が仕事以外には目も振れず、自由開発を楽しんでるということにも要因があるらしい。そんなしわくちゃの服を着た怪物たちが、一歩一歩成長していくところを見物する。とても嬉しいことだ。

技術者随時募集

C++文法サマリ

※〈　〉内は省略可能

Turbo C++	MIWA C++
■式	
式： 代入式 式,代入式	同左
代入式： 条件式 単項式　代入演算子　代入式	代入式： 式　代入演算子　式
代入演算子：以下のうちのひとつ =　*=　/=　%=　+=　-=　>>= <<=　&=　^=　|=	同左
条件式： 論理OR式 論理OR式　?　式　:　条件式	条件式： 式　?　式　:　式
論理OR式： 論理AND式 論理OR式　||　論理AND式	論理的OR式： 式　||　式
論理AND式： OR式 論理AND式　&&　OR式	論理的AND式： 式　&&　式
OR式： XOR式 OR式　|　XOR式	OR式 式　|　式
XOR式： AND式 XOR式　^　AND式	排他的OR式 式　^　式
AND式： 等価式 AND式　&　等価式	AND式 式　&　式
等価式： 関係式 等価式　==　関係式 等価式　!=　関係式	等価式： 式　==　式 式　!=　式
関係式： シフト式 関係式　<　シフト式 関係式　>　シフト式 関係式　<=　シフト式 関係式　>=　シフト式	関係式： 式　<　式 式　>　式 式　<=　式 式　>=　式
シフト式： 加減算式 シフト式　<<　加減算式 シフト式　>>　加減算式	シフト式： 式　<<　式 式　>>　式
加減算式： 乗除算式 加減算式　+　乗除算式 加減算式　-　乗除算式	加法式： 式　+　式 式　-　式
乗除算式： ポインタメンバ式 乗除算式　*　ポインタメンバ式 乗除算式　/　ポインタメンバ式 乗除算式　%　ポインタメンバ式	乗法式： 式　*　式 式　/　式 式　%　式
ポインタメンバ式： キャスト式 ポインタメンバ式　.*　キャスト式 ポインタメンバ式　->*　キャスト式	同左
キャスト式： 単項式 (型名)キャスト式	同左
単項式： 後置式 ++　単項式 --　単項式 単項演算子　キャスト式 sizeof 単項式 sizeof (型名) メモリ割り当て式 メモリ解放式	単項式： 単項演算子　式 式　++ 式　-- sizeof　式 sizeof (型名) (型名) 式 基本型名 (式の並び) new 型名 〈初期化子〉 new (型名) delete 式 delete [式] 式
単項演算子：以下のうちのひとつ *　&　+　-　!　~	単項演算子：以下のうちのひとつ *　&　+　-　!　~　++　--
メモリ割り当て式： 〈::〉 new 〈配置〉 制限型名 〈初期化子〉 〈::〉 new 〈配置〉 (型名)　〈初期化子〉	
配置： (式リスト)	
制限型名： 型指定子 〈制限宣言子〉	
制限宣言子： ポインタ演算子 〈制限宣言子〉 制限宣言子 [〈式〉]	
メモリ解放式： 〈::〉 delete キャスト式 〈::〉 delete [式] キャスト式	
後置式： 一次式 後置式 [式] 後置式 (〈式リスト〉) 単純型名 (〈式リスト〉) 後置式 .　名前 後置式 -> 名前 後置式 ++ 後置式 --	
式リスト： 代入式 式リスト,代入式	式の並び： 式 式の並び,式
一次式： リテラル 疑似変数 (式) this :: 識別子 :: 演算子関数名 名前	一次式： 識別 ::　識別子 定数 文字列 this (式) 一次式 [式] 一次式 (〈式の並び〉) 一次式 .　識別 一次式 ->　識別
	識別： 識別子 演算子関数名 typedef名　::　識別子 typedef名　::　演算子関数名
名前： 識別子 演算子関数名 変換関数名 限定名	
限定名： クラス名　::　識別子 クラス名　::　演算子関数名 クラス名　::　変換関数名 クラス名　::　クラス名 クラス名　::　~クラス名	
リテラル： 整数定数 文字定数 浮動小数点定数 文字列	
■宣言	
宣言： 〈宣言指定子〉〈宣言子リスト〉; アセンブラ宣言 関数宣言 リンケージ指定	宣言： 〈宣言指定子群〉〈宣言子並び〉; 名前宣言 asm宣言
宣言指定子： 記憶クラス指定子 型指定子 関数指定子 friend typedef	同左
	宣言指定子群： 宣言指定子 〈宣言指定子群〉
宣言指定子リスト： 〈宣言指定子リスト〉 宣言指定子	
記憶クラス指定子： auto static extern register	記憶クラス宣言子： auto register static extern
関数指定子： inline virtual	関数指定子： inline virtual overload
型指定子： 単純型名 クラス指定子 列挙指定子 厳密型指定子 const volatile	型指定子： 基本型名 クラス指定子 列挙指定子 複合型指定子 const volatile
単純型名： クラス 名 typedef 名 char short int long signed unsigned float double void	基本型名： typedef 名 char short int long unsigned float double void
厳密型指定子 クラスキーワード　識別子 クラスキーワード　クラス名 enum　列挙型名	複合型指定子： キー　typedef名 キー　識別子
クラスキーワード： class struct union	キー： class struct union enum
列挙指定子： enum 〈識別子〉 {〈列挙リスト〉}	列挙指定子： enum 〈識別子〉 {列挙の並び}
列挙リスト： 列挙子 列挙リスト　列挙子	列挙の並び： 列挙子 列挙の並び ,列挙子
列挙子： 識別子 識別子　=　定数式	同左
定数式： 条件式	
リンケージ指定： extern 文字列 {〈宣言リスト〉} extern 文字列　宣言	リンク指定： extern 文字列 {〈宣言リスト〉} extern 文字列　宣言

<table>
<tr><th>Turbo C++</th><th>MIWA C++</th></tr>
<tr><td>宣言リスト：
宣言
宣言リスト ； 宣言</td><td>同左</td></tr>
<tr><td></td><td>asm宣言：
asm（文字列）；</td></tr>
</table>

■宣言子

<table>
<tr><td>宣言子リスト：
初期化宣言子
宣言子リスト，初期化宣言子</td><td>宣言子並び：
初期宣言子
初期宣言子，宣言子並び</td></tr>
<tr><td>初期化宣言子：
宣言子〈初期化子〉</td><td>初期宣言子：
宣言子〈初期化子〉</td></tr>
<tr><td>宣言子：
宣言名
修飾子リスト
ポインタ演算子 宣言子
宣言子（引数宣言リスト）〈cv限定子リスト〉
宣言子［〈定数式〉］
（宣言子）</td><td>宣言子：
宣言名
（宣言子）
* 宣言子
* const 宣言子
& 宣言子
& const 宣言子
宣言子（引数宣言の並び）
宣言子［〈定数式〉］</td></tr>
<tr><td>修飾子リスト：
修飾子
修飾子リスト 修飾子</td><td></td></tr>
<tr><td>修飾子：
cdecl
pascal
interrupt
near
far
huge</td><td></td></tr>
<tr><td>ポインタ演算子：
* 〈cv 限定子リスト〉
& 〈cv 限定子リスト〉
クラス名 :: * 〈cv 限定子リスト〉</td><td></td></tr>
<tr><td>cv 限定子リスト：
cv 限定子〈cv 限定子リスト〉</td><td></td></tr>
<tr><td>cv 限定子：
const
volatile</td><td></td></tr>
<tr><td>宣言名：
名前
クラス名
~クラス名
typedef 名</td><td>宣言名：
基本宣言名
typedef 名
~typedef 名
演算子関数名
変換関数名</td></tr>
<tr><td>型名：
型指定子〈抽象宣言子〉</td><td>型名：
型指定子 抽象宣言子</td></tr>
<tr><td>抽象宣言子：
ポインタ演算子〈抽象宣言子〉
〈抽象宣言子〉（引数宣言リスト）〈cv限定子リスト〉
〈抽象宣言子〉［〈定数式〉］
（抽象宣言子）</td><td>抽象宣言子：
空
* 抽象宣言子
抽象宣言子（引数宣言の並び）
抽象宣言子［〈定数式〉］
（抽象宣言子）</td></tr>
<tr><td>引数宣言リスト：
〈arg-宣言リスト〉
arg-宣言リスト ，・・・
〈arg-宣言リスト〉・・・</td><td>引数宣言の並び：
〈引数宣言並び〉
〈引数宣言並び〉・・・</td></tr>
<tr><td>arg-宣言リスト：
引数宣言
arg-宣言リスト ，引数宣言</td><td>引数宣言並び：
引数宣言並び ， 引数宣言
引数宣言</td></tr>
<tr><td>引数宣言：
宣言指定子リスト 宣言子
宣言指定子リスト 宣言子 ＝ 式
宣言指定子リスト〈抽象宣言子〉
宣言指定子リスト〈抽象宣言子〉＝ 式</td><td>引数宣言：
宣言指定子群 宣言子
宣言指定子群 宣言子 ＝ 式
宣言指定子群 抽象宣言子
宣言指定子群 抽象宣言子 ＝ 式</td></tr>
<tr><td>関数定義：
〈宣言指定子リスト〉宣言子〈コンストラクタ初期化子〉関数本体</td><td>関数定義：
〈宣言指定子群〉関数宣言子〈基底初期化子〉
関数本体</td></tr>
<tr><td>関数本体：
複合文</td><td>関数本体：
複文</td></tr>
<tr><td>初期化子：
＝ 式
＝｛初期化子リスト｝
（式リスト）</td><td>初期化子：
＝ 式
＝｛初期化子の並び｝
＝｛初期化子の並び，｝
｛式の並び｝</td></tr>
<tr><td>初期化子リスト：
式
初期化子リスト ，式
｛初期化子リスト〈,〉｝</td><td>初期化子の並び：
式
初期化子の並び ，初期化子の並び
｛初期化子の並び｝</td></tr>
</table>

■クラス宣言

<table>
<tr><td>クラス指定子：
クラス頭部｛〈メンバリスト〉｝</td><td>クラス指定子：
クラス頭｛〈メンバ並び〉｝</td></tr>
<tr><td>クラス頭部：
クラスキーワード〈識別子〉〈基本型指定子〉
クラスキーワード クラス名〈基本型指定子〉</td><td>クラス頭：
集合名〈識別子〉
集合名 識別子 ： typedef名
集合名 識別子 ： public typedef名</td></tr>
<tr><td>クラスキーワード：
class
struct
union</td><td>集合名：
class
struct
union</td></tr>
<tr><td>メンバリスト：
メンバ宣言 〈メンバリスト〉
アクセス指定子 ：〈メンバリスト〉</td><td>メンバの並び：
メンバ宣言 〈メンバの並び〉</td></tr>
<tr><td>メンバ宣言：
〈宣言指定子〉〈メンバ宣言リスト〉；
関数定義〈;〉
限定名 ；</td><td>メンバ宣言：
〈宣言指定子群〉 メンバ宣言 ；
関数定義
関数定義 ；</td></tr>
<tr><td>メンバ宣言リスト：
メンバ宣言
メンバ宣言 ，メンバ宣言</td><td></td></tr>
<tr><td>メンバ宣言：
宣言子〈純粋指定子〉
〈識別子〉： 定数式</td><td>メンバ宣言子：
宣言子〈純粋指定子〉
〈識別子〉： 定数式</td></tr>
<tr><td>純粋指定子：
＝ 0</td><td>同左</td></tr>
<tr><td>基本クラス指定：
： 基本クラスリスト</td><td></td></tr>
<tr><td>基本クラスリスト：
基本クラス指定子
基本クラスリスト ，基本クラス指定子</td><td></td></tr>
<tr><td>基本クラス指定子：
クラス名
virtual〈アクセス指定子〉クラス名
アクセス指定子〈virtual〉クラス名</td><td></td></tr>
<tr><td>アクセス指定子：
private
protected
public</td><td>メンバ列：
private：メンバの並び
protected：メンバの並び
public：メンバの並び</td></tr>
<tr><td>変換関数名：
operator 変換型名</td><td>変換関数名：
operator 型名</td></tr>
<tr><td>変換型名：
型指定子〈ポインタ演算子〉</td><td></td></tr>
<tr><td>コンストラクタ初期化子：
：メンバ初期化子リスト</td><td>基底初期化子：
：メンバ初期化子の並び</td></tr>
<tr><td>メンバ初期化子リスト：
メンバ初期化子
メンバ初期化子 ，メンバ初期化子リスト</td><td>メンバ初期化子の並び：
メンバ初期化子
メンバ初期化子 ，メンバ初期化子の並び</td></tr>
<tr><td>メンバ初期化子：
クラス名（＜引数リスト＞）
識別子（＜引数リスト＞）</td><td>メンバ初期化子：
〈識別子〉（＜引数の並び＞）</td></tr>
<tr><td>演算子関数名：
operator 演算子</td><td>同左</td></tr>
<tr><td>演算子：以下のうちのひとつ
new delete sizeof
+ - * / % ^ & | ~
! = < > += -= *= /= %=
^= &= |= << >> >>= <<= == !=
<= >= && || ++ -- , ->* ->
() []</td><td>演算子：以下のうちのひとつ
new delete
+ - * / % ^ & | ~
! = < > += -= *= /= %=
^= &= |= << >> >>= <<= == !=
<= >= && || ++ -- , ->* ->
() []</td></tr>
</table>

■文

<table>
<tr><td>文：
ラベルつき文
複合文
式文
選択文
繰り返し文
ジャンプ文
アセンブラ文
宣言</td><td></td></tr>
<tr><td>アセンブラ文：
asm トークン 改行
asm トークン ；
asm ｛ トークン；〈トークン；〉＝
〈トークン；〉
｝</td><td></td></tr>
<tr><td>ラベルつき文：
識別子 ： 文
case 定数式 ： 文
default ： 文</td><td>名札つき文：
識別子 ： 文
switch 文 ：</td></tr>
<tr><td></td><td>switch（式）文
case 定数式 ：
default ：</td></tr>
<tr><td>式文：
〈式〉；</td><td>式の文：
式；</td></tr>
<tr><td>複合文：
｛〈宣言リスト〉〈文リスト〉｝</td><td>複文：
｛〈文の並び〉｝</td></tr>
<tr><td>宣言リスト：
宣言
宣言リスト 宣言</td><td></td></tr>
<tr><td>文リスト：
文
文リスト 文</td><td>文の並び：
文
文 文の並び</td></tr>
<tr><td>選択文：
if（式）文
if（式）文 else 文
switch（式）文</td><td>条件文：
if（式）文
if（式）文 else 文</td></tr>
<tr><td>繰り返し文：
while（式）文
do 文 while（式）；
for（for初期化文;〈式〉;〈式〉）文</td><td>while 文：while（式）文
do 文：do 文 while（式）；
for 文：for（〈文1〉〈式1〉；〈式2〉）〈文2〉</td></tr>
<tr><td>for初期化文：
式文
宣言</td><td></td></tr>
<tr><td>ジャンプ文：
goto 識別子 ；
continue ；
break ；
return〈式〉；</td><td>break 文；break ；
continue 文：continue ；
return 文：return ；
return 式 ；
goto 文：goto 識別子</td></tr>
<tr><td>宣言文：
宣言</td><td>同左</td></tr>
</table>

COMPUTER PRO-STAFF
SYSTEM CREATE



すべてのCプログラマのために……

プロジェクト・マネージャーを強力にサポートします。

C言語開発支援ツール

CPAD Version 2

シーパッド Ver.2

98シリーズ対応版…… ￥85,000(税抜)
ダブルメディア 3.5、5インチ添付

高品質のドキュメンテーションを驚くほど簡単に実現する、新世代のプロジェクト指向グループウェア「CPAD V.2」登場。

バージョンアップ 機能強化

CPAD Ver1.5ご購入登録ユーザ様は無償バージョンアップ(送料のみご負担いただきます)

「CPAD」はこれまでC言語による開発でソフトハウスが悩んでいた開発時の工数短縮及び複数人数によるプロジェクト管理を容易にいたします。また、既存のソースリストの仕様書を自動作成しますので、管理体制を容易に画一化することが可能となり、プログラマのドキュメンテーションの負担を軽減することができます。

6大機能 Cソースの解析、援助、学習ツールとして

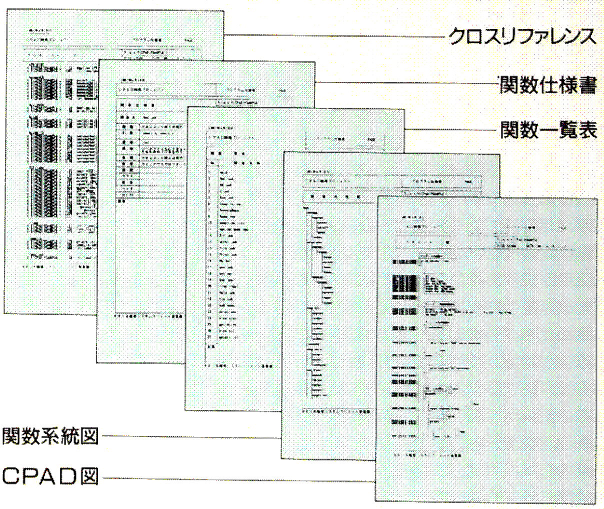
①C言語ソースのドキュメント管理を自動化、一律化
②全ドキュメントはソース上で統括管理
③CPAD統合環境モード新規登載
④カーニハン&リッチーの書式にソースをリフォーム
⑤PAD様式に準ずる出力機能
⑥エディタ編集可能なドキュメントと印刷スタイル設定機能

動作環境

①対応パソコン	NEC PC9801シリーズ、EPSON PC-386・PC-286シリーズ
②対応OS	MS-DOS V2.11以上
③メモリ	640Kバイト以上
④周辺機器	NEC 201サポートのプリンタ、ハードディスク、RAMディスクなど MS-DOS上で利用可能なメディアであれば、何れも可
⑤ディスプレイ	640×400であれば、何れも可。
⑥対応C言語	ANSI準拠によるC言語コンパイラ用ソースであれば、何れも可。

● MS-DOSは米国マイクロソフト社の商標です。(本製品にはMS-DOSは含まれておりません)

自動出力印刷例



※お求めの際は有名パソコンショップ、または右記口座にお振り込みのうえ、電話にてご注文下さい。太陽神戸三井銀行 難波支店 普通口座 5221100 ユ)オオツカショウジシステムクリエイト

オオツカ商事 COMPUTER PRO-STAFF SYSTEM CREATE
本社:〒760 高松市西内町5-14オオツカビル内

ソフトウェア インフォメーションセンター(技術的なご質問・ご相談に電話でお答えします)
TEL.(0878)51-1807
受付時間 AM10:00〜PM5:00 月曜日〜金曜日(祝日を除く)

外部スタッフ、及び正社員募集中!! 近距離、遠距離を問わず 詳しくは人事部直通電話(0878)51-1785まで

<資料請求番号P03>

T4911777708982

〈資料請求番号P04〉

雑誌コード17777-08